In [356…

```
%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 from data that was collected from 01/01/2020 to 01/10/2020.

Import dataset.

In [205…

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

Out[205…

	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…
```
df.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
```

In [132…
```
df['cases'].describe()
```

Out[132…
```
count    57750.000000
mean       590.919827
std       4068.854106
min          0.000000
25%          0.000000
50%          1.000000
```

```
75%          83.000000
max       97894.000000
Name: cases, dtype: float64
```

Data Cleaning

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

In [206…

```
df=df.replace(np.nan, 0)
df=df.drop('geoid', axis=1)
df=df.drop('iso', axis=1)
df=df.drop('country', axis=1)
df=df.drop('continent', axis=1)
df=df.drop('d', axis=1)
df=df.drop('deaths', axis=1)
df=df.drop('population_2019', axis=1)
df=df.drop('school_local', axis=1)
df=df.drop('domestic_local', axis=1)
df=df.drop('travel_partial', axis=1)
df=df.drop('travel_dom_partial', axis=1)
df=df.drop('curf_partial', axis=1)
df=df.drop('mass_partial', axis=1)
df=df.drop('elect_partial', axis=1)
df=df.drop('sport_partial', axis=1)
df=df.drop('rest_local', axis=1)
df=df.drop('testing_narrow', axis=1)
df=df.drop('masks_partial', axis=1)
df=df.drop('surveillance_partial', axis=1)
df=df.drop('state_partial', axis=1)
df=df.drop('Rigidity_Public_Health', axis=1)
df=df.drop('Economic_Measures', axis=1)
df
```

Out[206…

	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 logit model will be determining whether cases of the virus were present, two classes. Makes all values where cases were above 0 into 1. The XGBoost model will be determining whether cases of the virus were present from three classes, which are small, medium, and large amount of cases.

In [207…
```
cases_lr=df['cases'].copy()
cases_lr.values[cases_lr > 0] = 1
```

In [215…
```
cases_xgb=pd.qcut(df['cases'], q=5, duplicates='drop')
```

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 :

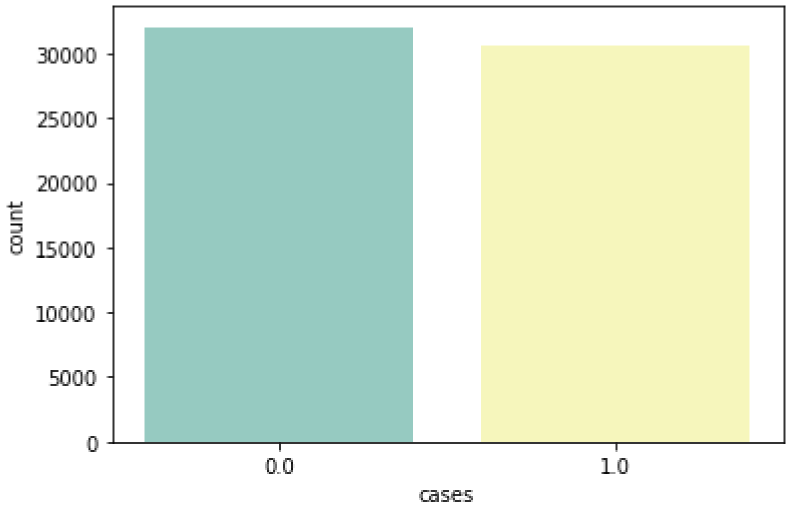
- cases: binary variable equal to 1 if there were cases of SARS-CoV-2 and 0 otherwise;
- school: binary variable equal to 1 if schools were closed and 0 otherwise;
- domestic: binary variable equal to 1 if there was a domestic lockdown and 0 otherwise;
- travel: binary variable equal to 1 if travel restrictions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

Oversampling will be used to resolve class imbalances.

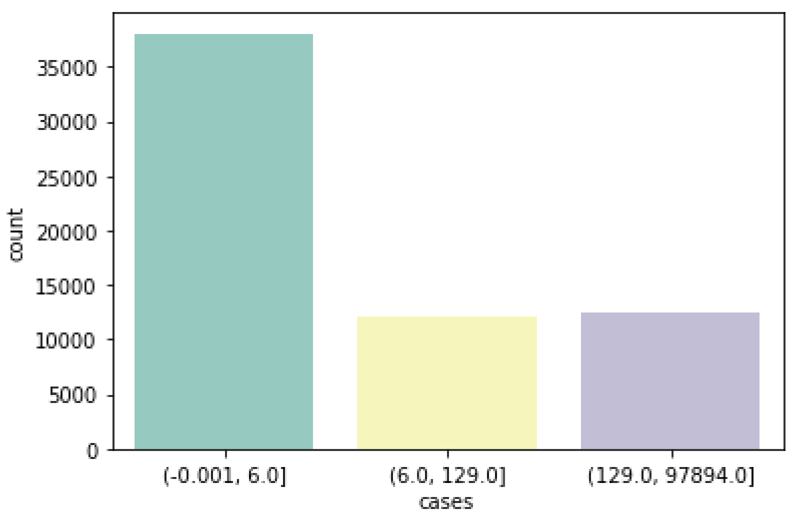
In [211…
```
sns.countplot(cases_lr, palette='Set3')
```

Out[211…
```
<AxesSubplot:xlabel='cases', ylabel='count'>
```



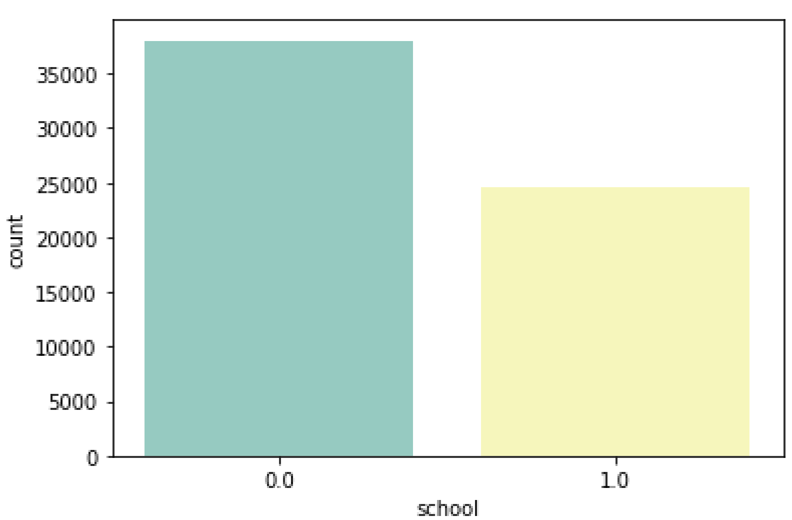
In [212…
```
sns.countplot(cases_gb, palette='Set3')
```

Out[212… <AxesSubplot:xlabel='cases', ylabel='count'>



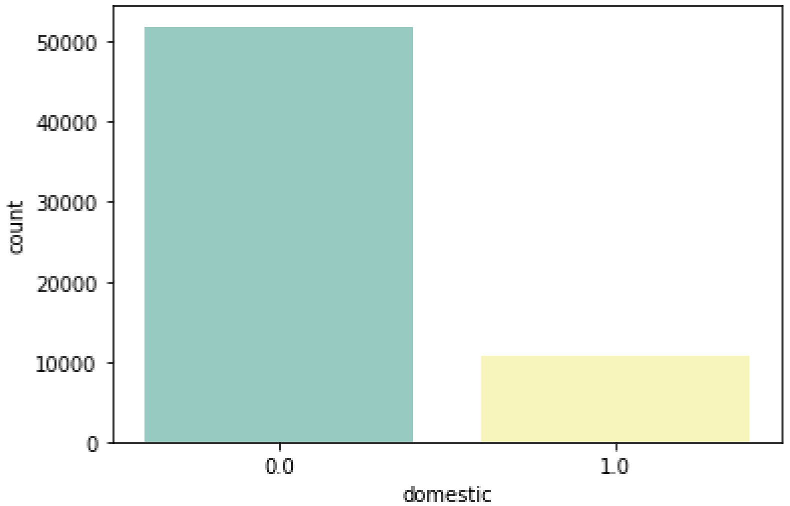
In [34]:
```
sns.countplot(df['school'], palette='Set3')
```

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



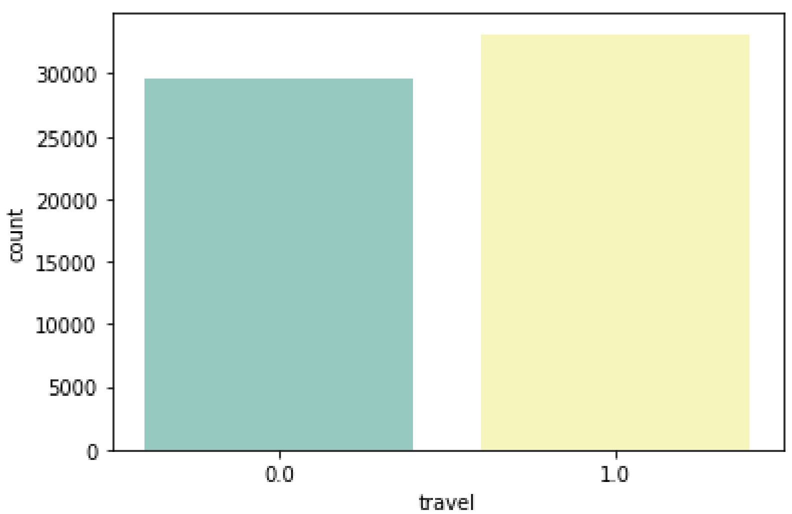
In [43]:
```
sns.countplot(df['domestic'], palette='Set3')
```

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



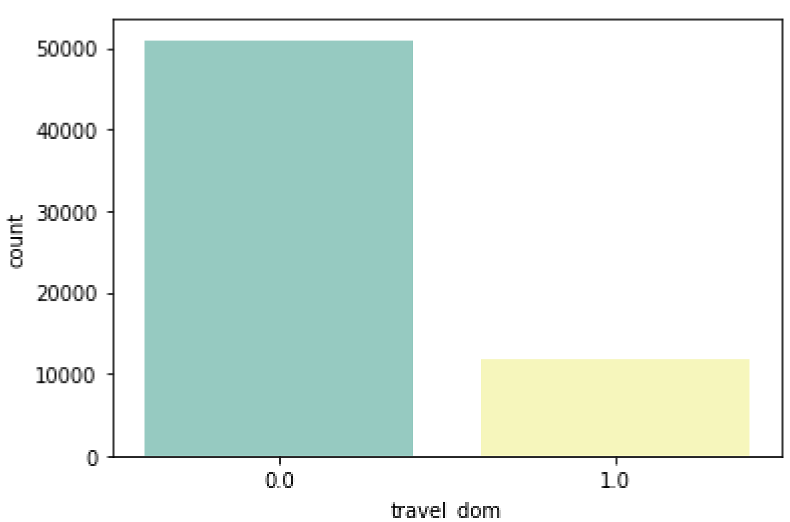
In [44]:
```
sns.countplot(df['travel'], palette='Set3')
```

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

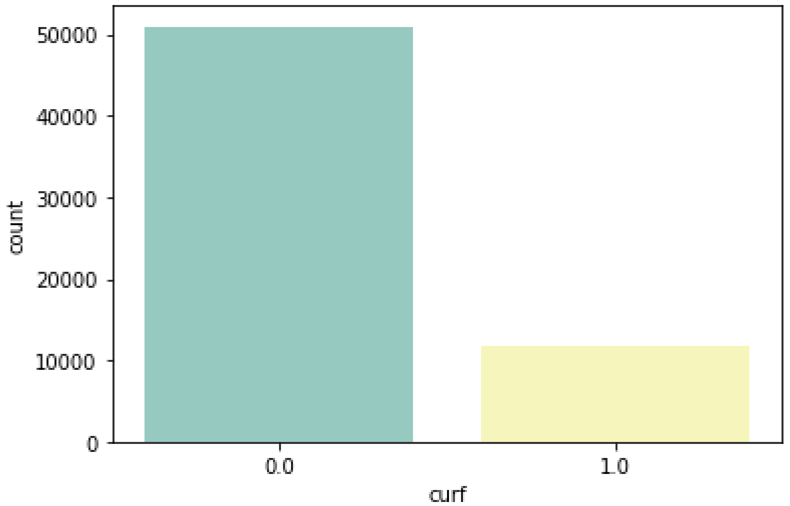


In [45]:
```
sns.countplot(df['travel_dom'], palette='Set3')
```

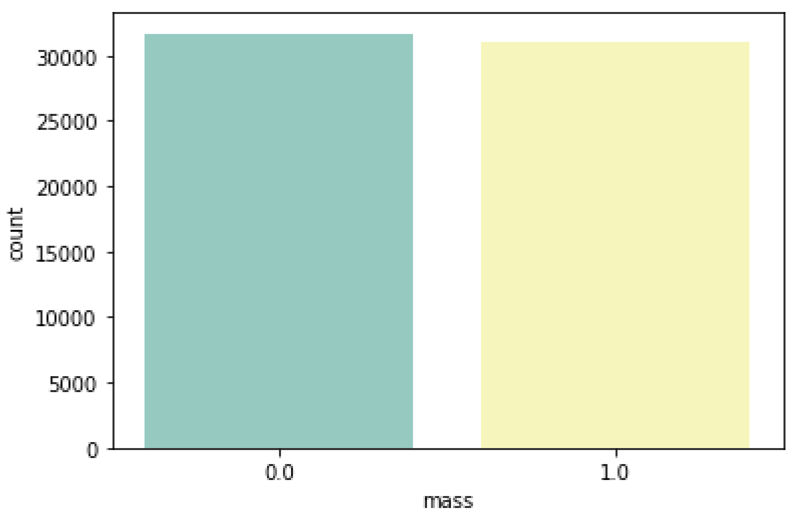
Out[45]: <AxesSubplot:xlabel='travel_dom', ylabel='count'>



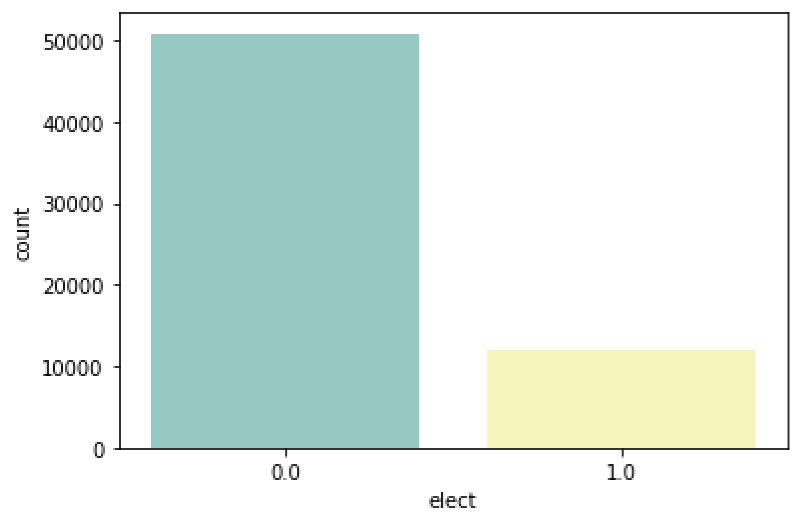
In [46]:
```
sns.countplot(df['curf'], palette='Set3')
```
Out[46]: <AxesSubplot:xlabel='curf', ylabel='count'>



In [47]:
```
sns.countplot(df['mass'], palette='Set3')
```
Out[47]: <AxesSubplot:xlabel='mass', ylabel='count'>

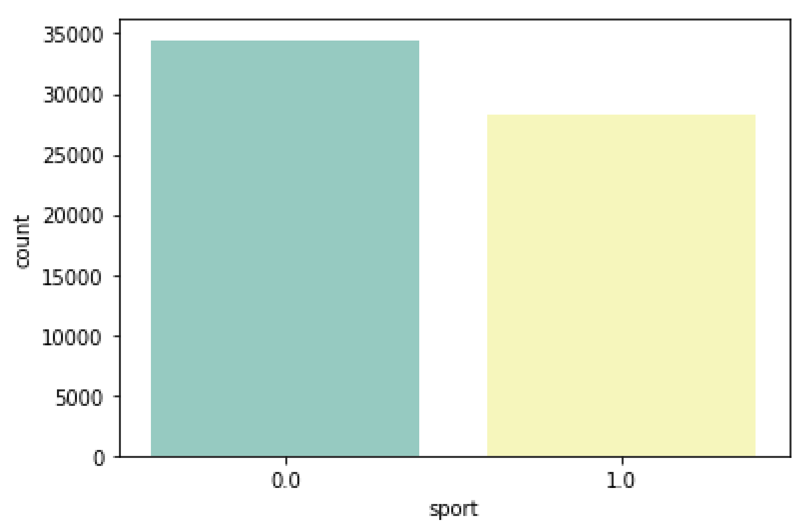


In [48]:
```
sns.countplot(df['elect'], palette='Set3')
```
Out[48]: <AxesSubplot:xlabel='elect', ylabel='count'>



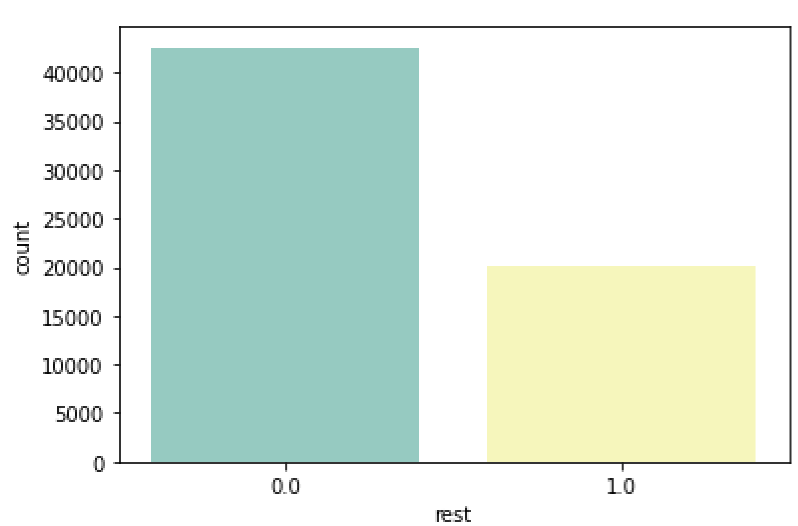
In [49]:
```
sns.countplot(df['sport'], palette='Set3')
```

Out[49]: <AxesSubplot:xlabel='sport', ylabel='count'>



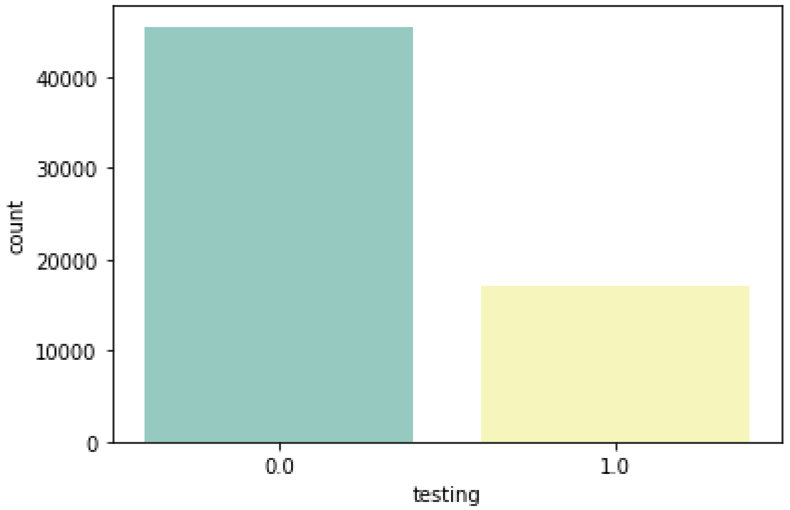
In [50]:
```
sns.countplot(df['rest'], palette='Set3')
```

Out[50]: <AxesSubplot:xlabel='rest', ylabel='count'>



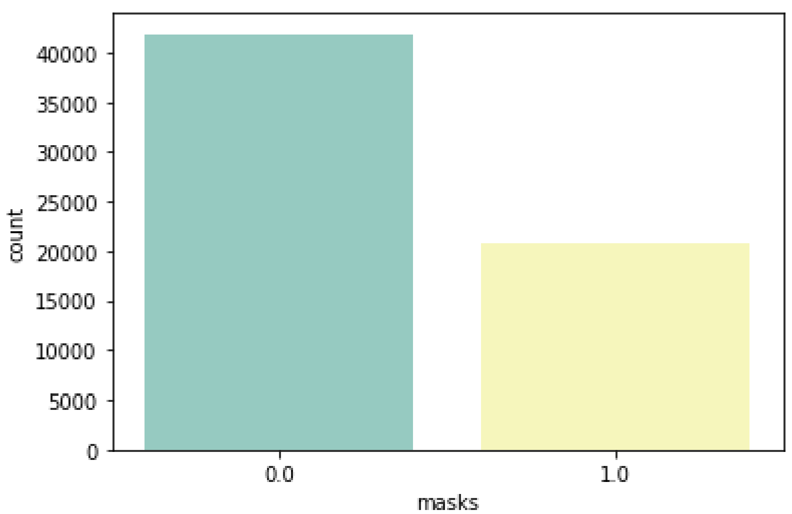
In [51]:
```
sns.countplot(df['testing'], palette='Set3')
```

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



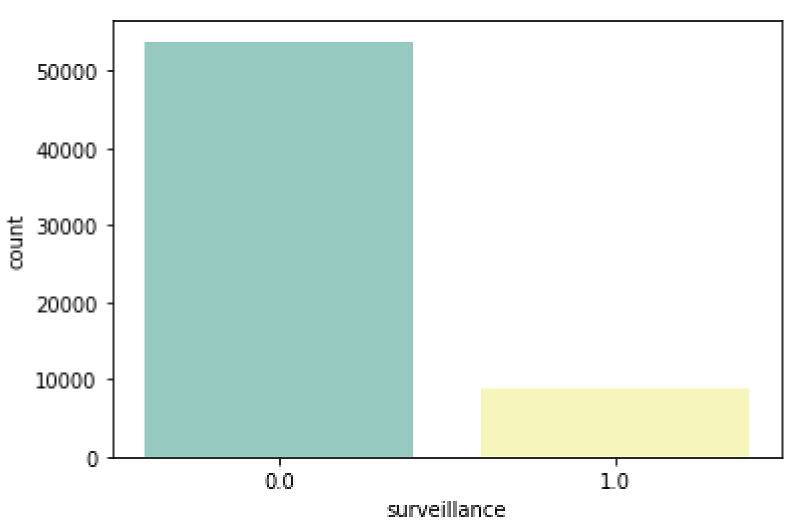
In [52]:
```
sns.countplot(df['masks'], palette='Set3')
```

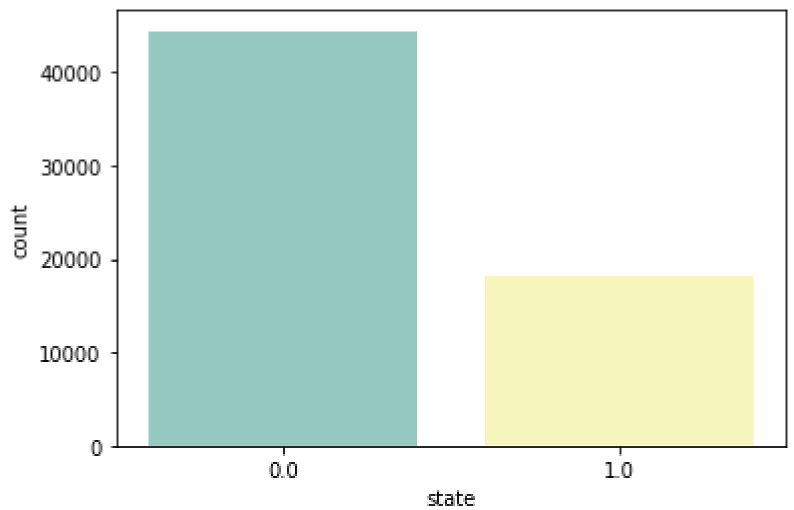
Out[52]: <AxesSubplot:xlabel='masks', ylabel='count'>

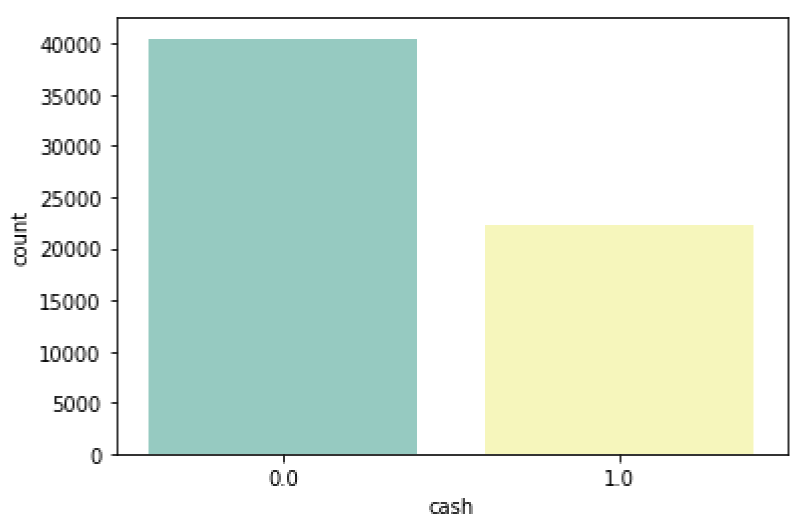


In [53]:
```
sns.countplot(df['surveillance'], palette='Set3')
```

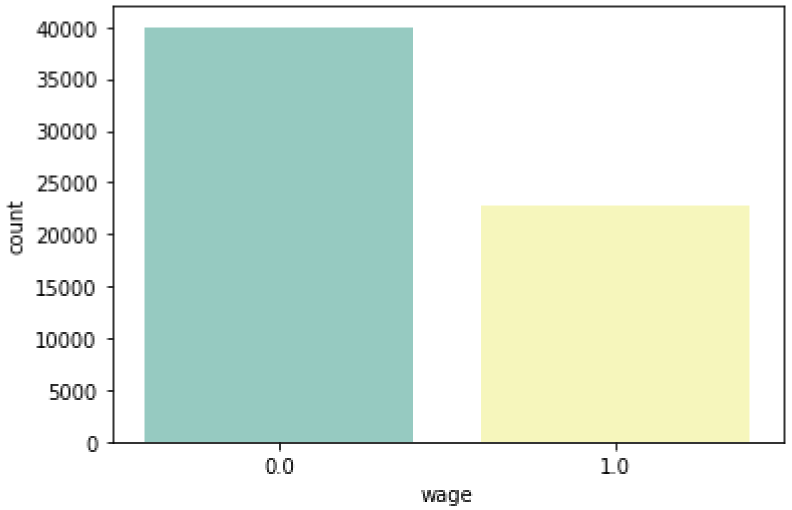
Out[53]: <AxesSubplot:xlabel='surveillance', ylabel='count'>



In [54]:
```
sns.countplot(df['state'], palette='Set3')
```
Out[54]:
```
<AxesSubplot:xlabel='state', ylabel='count'>
```


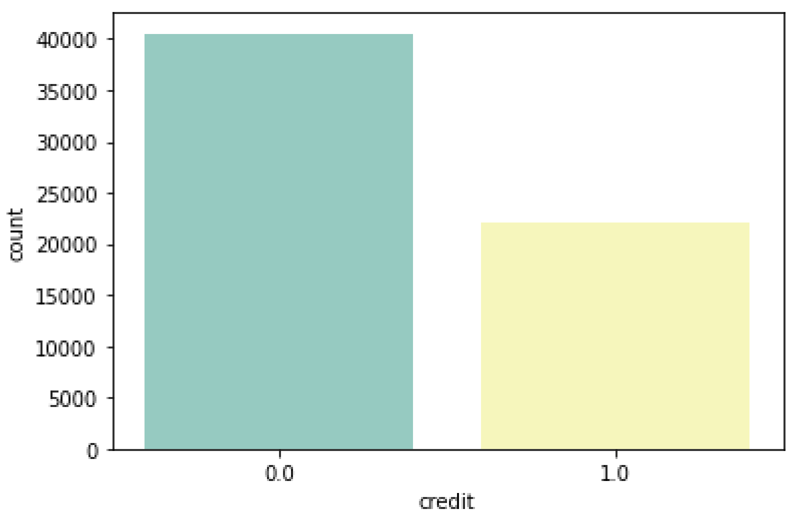
In [55]:
```
sns.countplot(df['cash'], palette='Set3')
```
Out[55]:
```
<AxesSubplot:xlabel='cash', ylabel='count'>
```


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



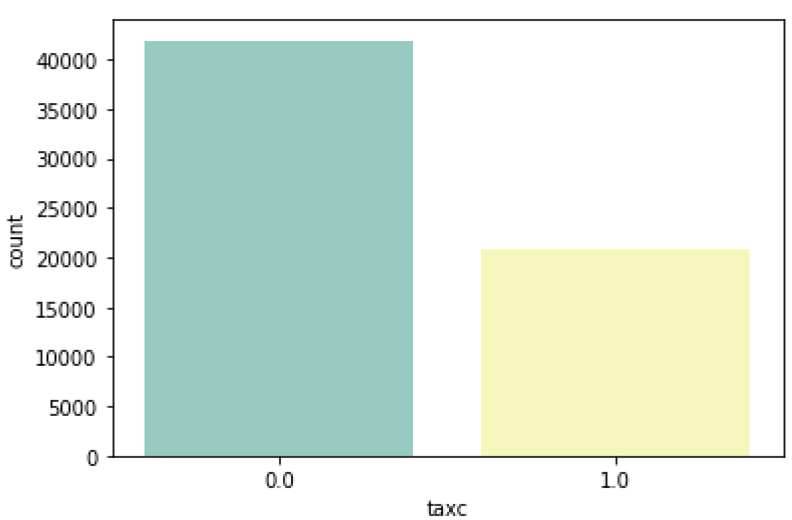
In [57]:
```
sns.countplot(df['credit'], palette='Set3')
```

Out[57]: <AxesSubplot:xlabel='credit', ylabel='count'>



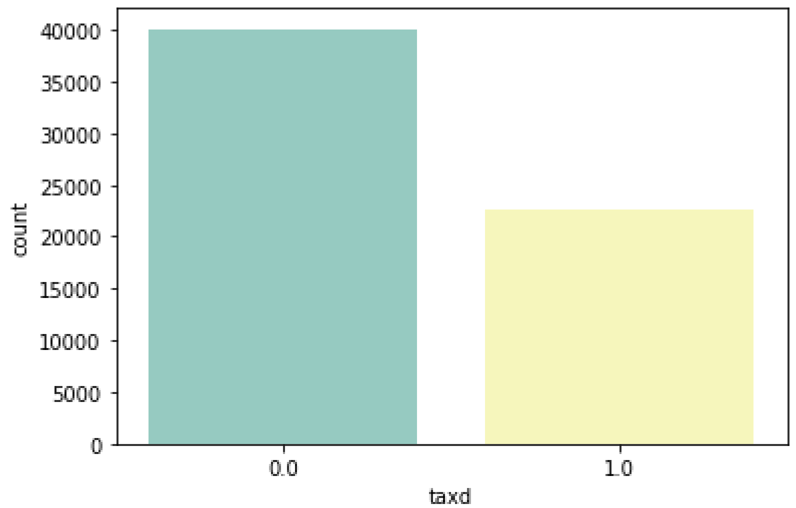
In [58]:
```
sns.countplot(df['taxc'], palette='Set3')
```

Out[58]: <AxesSubplot:xlabel='taxc', ylabel='count'>



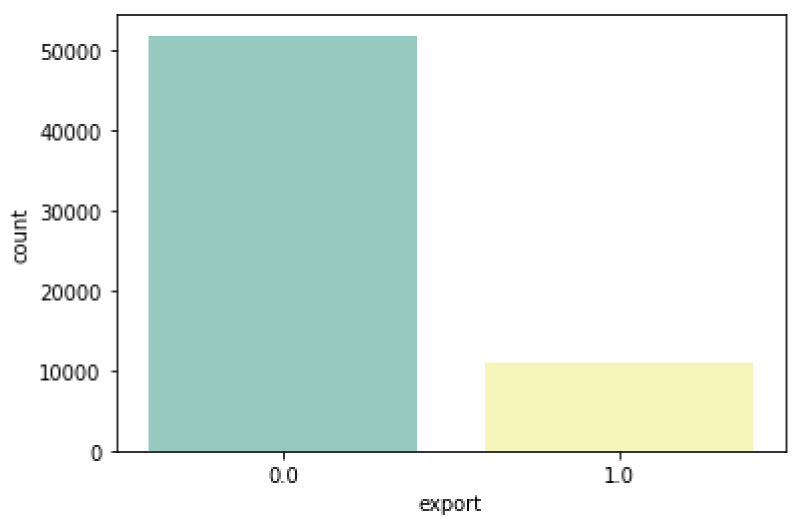
In [59]:
```
sns.countplot(df['taxd'], palette='Set3')
```

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



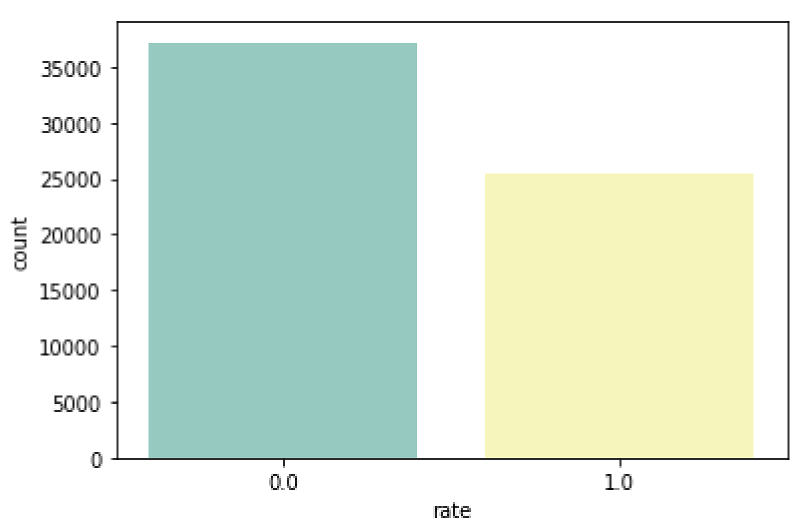
In [60]:
```
sns.countplot(df['export'], palette='Set3')
```

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



In [61]:
```
sns.countplot(df['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 [216…
```
y=cases_lr
X=df.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 [217…
```
ce_target = ce.TargetEncoder()

ce_target.fit(X, y)

ce_target.transform(X, y)
```

Out[217…

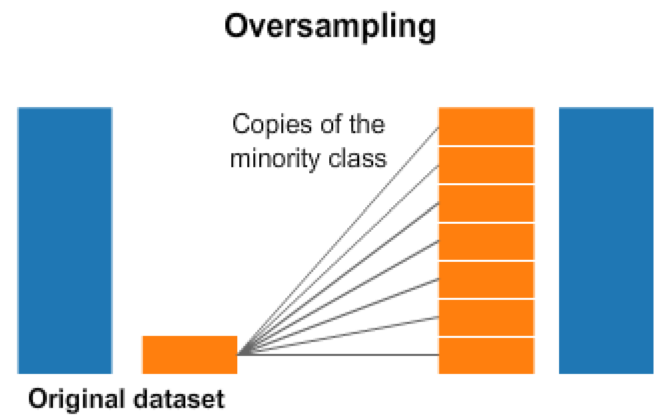
	school	domestic	travel	travel_dom	curf	mass	elect	sport	rest	testing	masks	surveillance	state	cash	wage	credit	taxc	taxd	export	rate
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	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
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

Oversampling selects data from the population minority class to be a larger amount of the sample.

In [218…
```
Image(filename='oversampling.png')
```

Out[218…



In [219…:

```
oversample = RandomOverSampler(sampling_strategy='minority')
X_resampled, y_resampled = oversample.fit_resample(X, y)
```

In [220…:

```
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

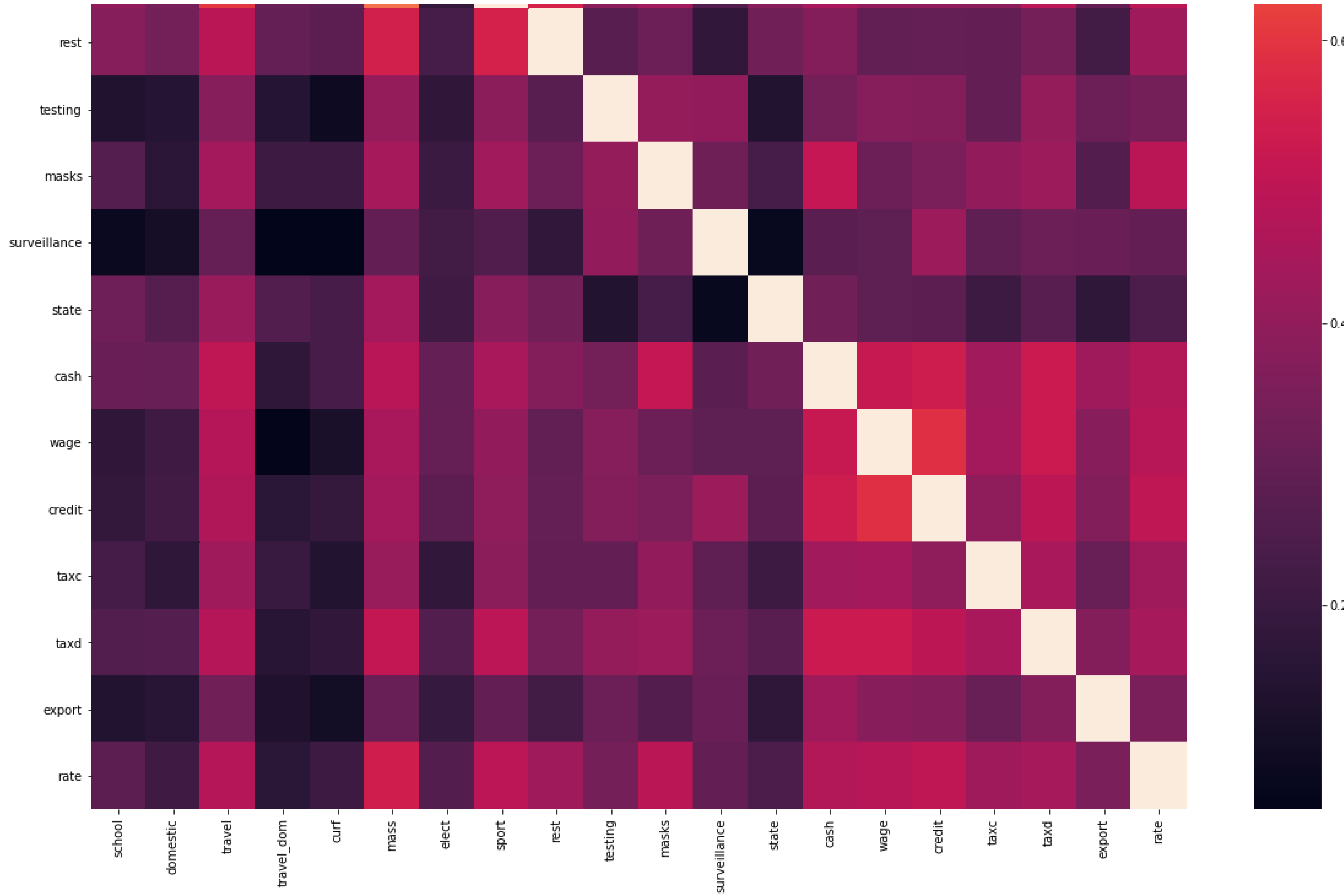
Binary logistic regression assumsions are a binary dependent variable, independent observations, linear continuous variables, no strongly influential outliers, independent variables with no multicollinearity, and and a large sample size. The dependent variable is zero or one, the observations are not 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 [221…

```
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.

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

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

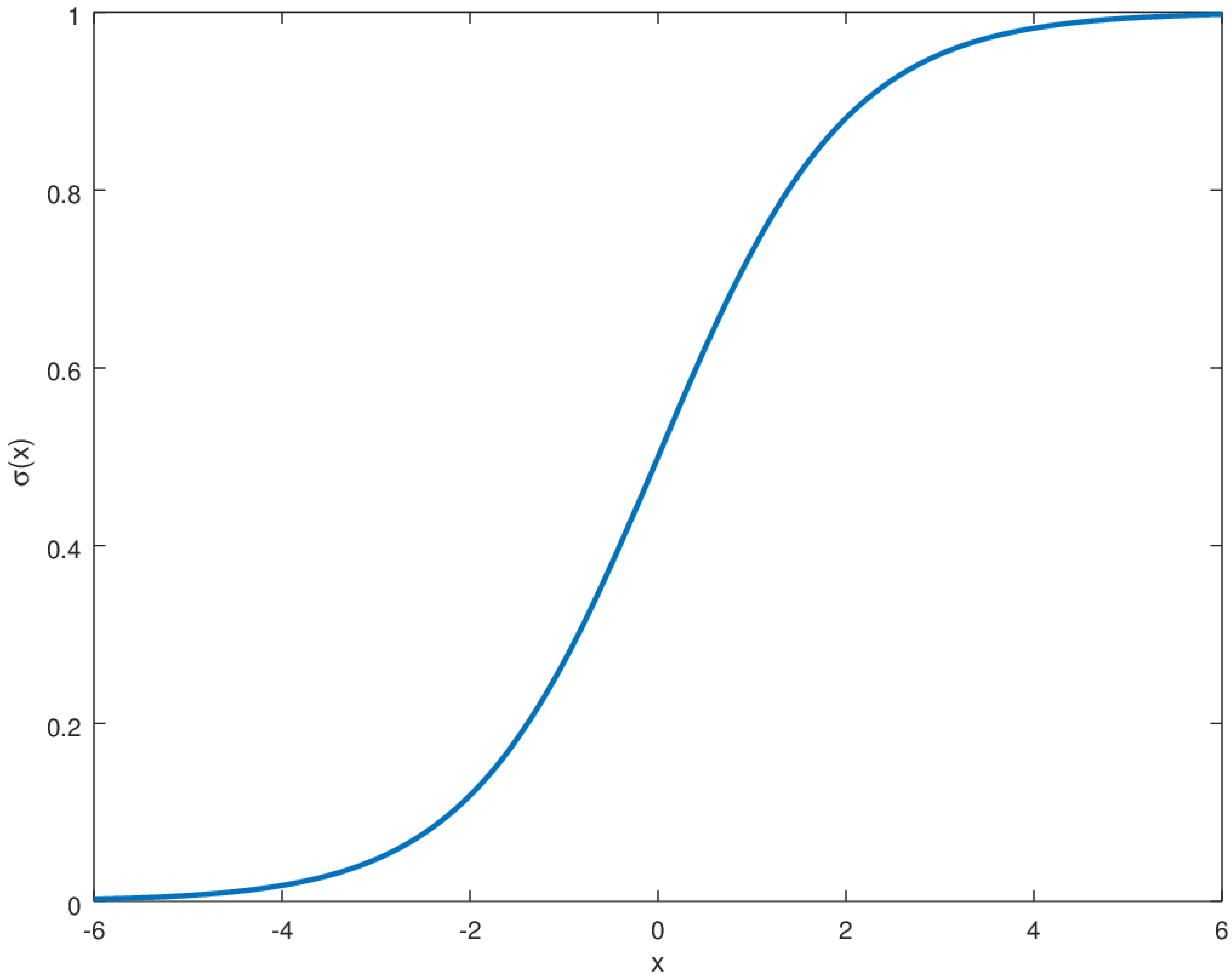
Take the inverse of P to get the desired sigmoid function.

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

In [27]:

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

Out[27]:

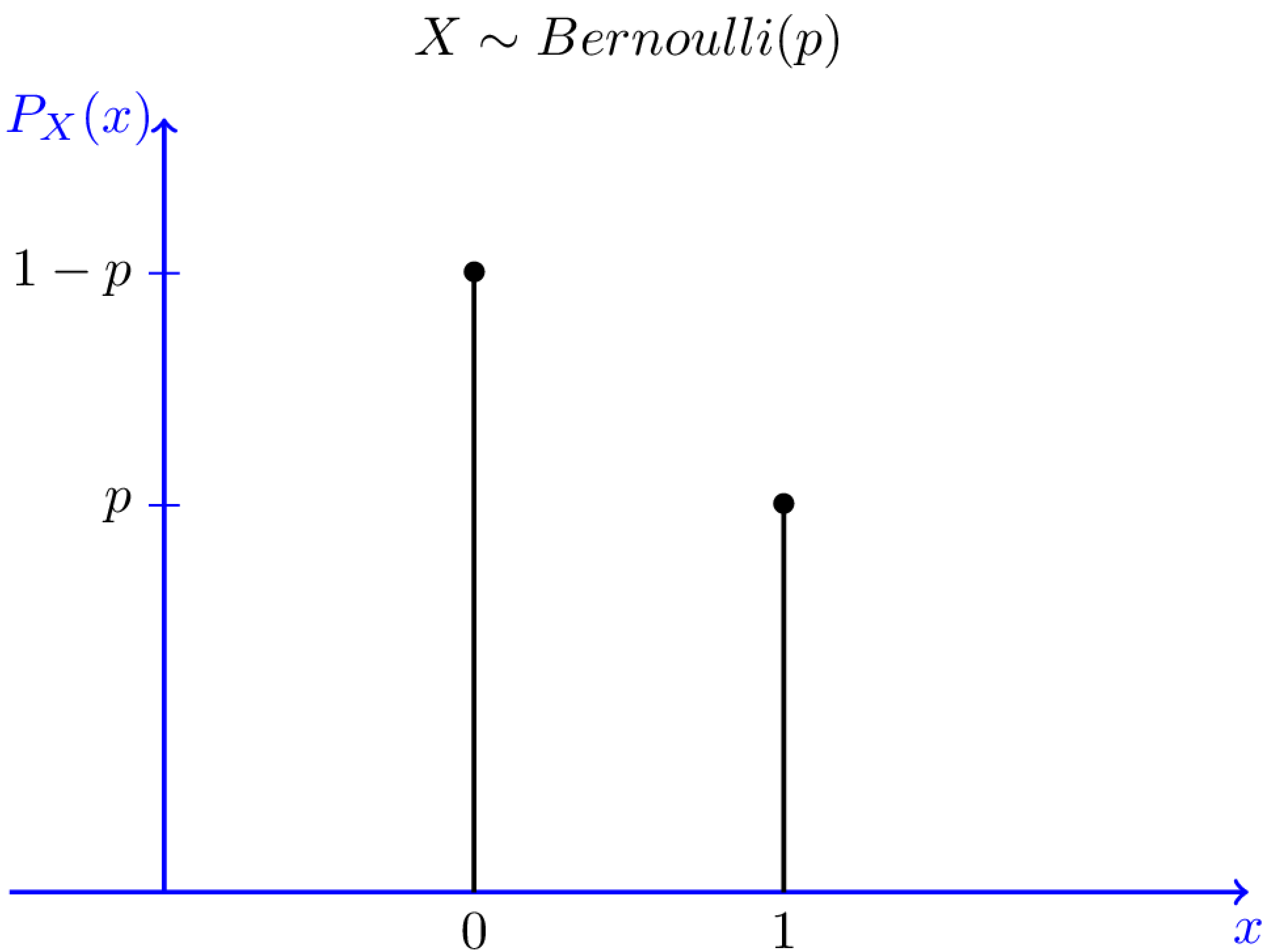


The predicted labels are binary so each label is a Bernoulli random variable from a Bernoulli probability mass function.

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

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.

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

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

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 [222…

```
log_reg = sm.Logit(y_train, X_train).fit()
log_reg.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.567829
         Iterations 6
```

Out[222…

Logit Regression Results

Dep. Variable:	cases	**No. Observations:**	51252
Model:	Logit	**Df Residuals:**	51233
Method:	MLE	**Df Model:**	18
Date:	Wed, 07 Apr 2021	**Pseudo R-squ.:**	0.1808
Time:	17:55:25	**Log-Likelihood:**	-29102.
converged:	True	**LL-Null:**	-35525.
Covariance Type:	nonrobust	**LLR p-value:**	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
school	-0.2738	0.025	-10.936	0.000	-0.323	-0.225
domestic	0.5573	0.035	15.702	0.000	0.488	0.627
travel	-0.4869	0.031	-15.630	0.000	-0.548	-0.426
travel_dom	0.0907	0.033	2.748	0.006	0.026	0.155
curf	0.1588	0.032	4.989	0.000	0.096	0.221
mass	0.3782	0.032	11.733	0.000	0.315	0.441
elect	0.5211	0.032	16.216	0.000	0.458	0.584
rest	0.4596	0.030	15.343	0.000	0.401	0.518
testing	-0.0358	0.030	-1.200	0.230	-0.094	0.023

masks	0.7856	0.030	25.992	0.000	0.726	0.845
surveillance	0.3829	0.041	9.360	0.000	0.303	0.463
state	-0.1365	0.028	-4.909	0.000	-0.191	-0.082
cash	0.7242	0.032	22.497	0.000	0.661	0.787
wage	-0.0997	0.032	-3.119	0.002	-0.162	-0.037
credit	0.0669	0.032	2.093	0.036	0.004	0.129
taxc	-0.3687	0.029	-12.683	0.000	-0.426	-0.312
taxd	0.4683	0.030	15.393	0.000	0.409	0.528
export	0.2075	0.036	5.845	0.000	0.138	0.277
rate	-0.0736	0.029	-2.518	0.012	-0.131	-0.016

Remove statistically insignificant feature.

In [224…
```
X_train=X_train.drop('testing', axis=1)
```

In [225…
```
log_reg = sm.Logit(y_train, X_train)
model=log_reg.fit()
model.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.567843
         Iterations 6
```

Out[225…

Logit Regression Results

Dep. Variable:	cases	**No. Observations:**	51252
Model:	Logit	**Df Residuals:**	51234
Method:	MLE	**Df Model:**	17
Date:	Wed, 07 Apr 2021	**Pseudo R-squ.:**	0.1808
Time:	17:55:47	**Log-Likelihood:**	-29103.
converged:	True	**LL-Null:**	-35525.
Covariance Type:	nonrobust	**LLR p-value:**	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
school	-0.2723	0.025	-10.888	0.000	-0.321	-0.223
domestic	0.5582	0.035	15.736	0.000	0.489	0.628
travel	-0.4899	0.031	-15.770	0.000	-0.551	-0.429
travel_dom	0.0892	0.033	2.703	0.007	0.025	0.154
curf	0.1603	0.032	5.042	0.000	0.098	0.223
mass	0.3737	0.032	11.669	0.000	0.311	0.436
elect	0.5209	0.032	16.211	0.000	0.458	0.584
rest	0.4567	0.030	15.292	0.000	0.398	0.515
masks	0.7794	0.030	26.173	0.000	0.721	0.838
surveillance	0.3744	0.040	9.291	0.000	0.295	0.453

state	-0.1342	0.028	-4.838	0.000	-0.189	-0.080
cash	0.7254	0.032	22.543	0.000	0.662	0.788
wage	-0.1025	0.032	-3.216	0.001	-0.165	-0.040
credit	0.0655	0.032	2.051	0.040	0.003	0.128
taxc	-0.3679	0.029	-12.657	0.000	-0.425	-0.311
taxd	0.4634	0.030	15.374	0.000	0.404	0.522
export	0.2038	0.035	5.762	0.000	0.134	0.273
rate	-0.0729	0.029	-2.497	0.013	-0.130	-0.016

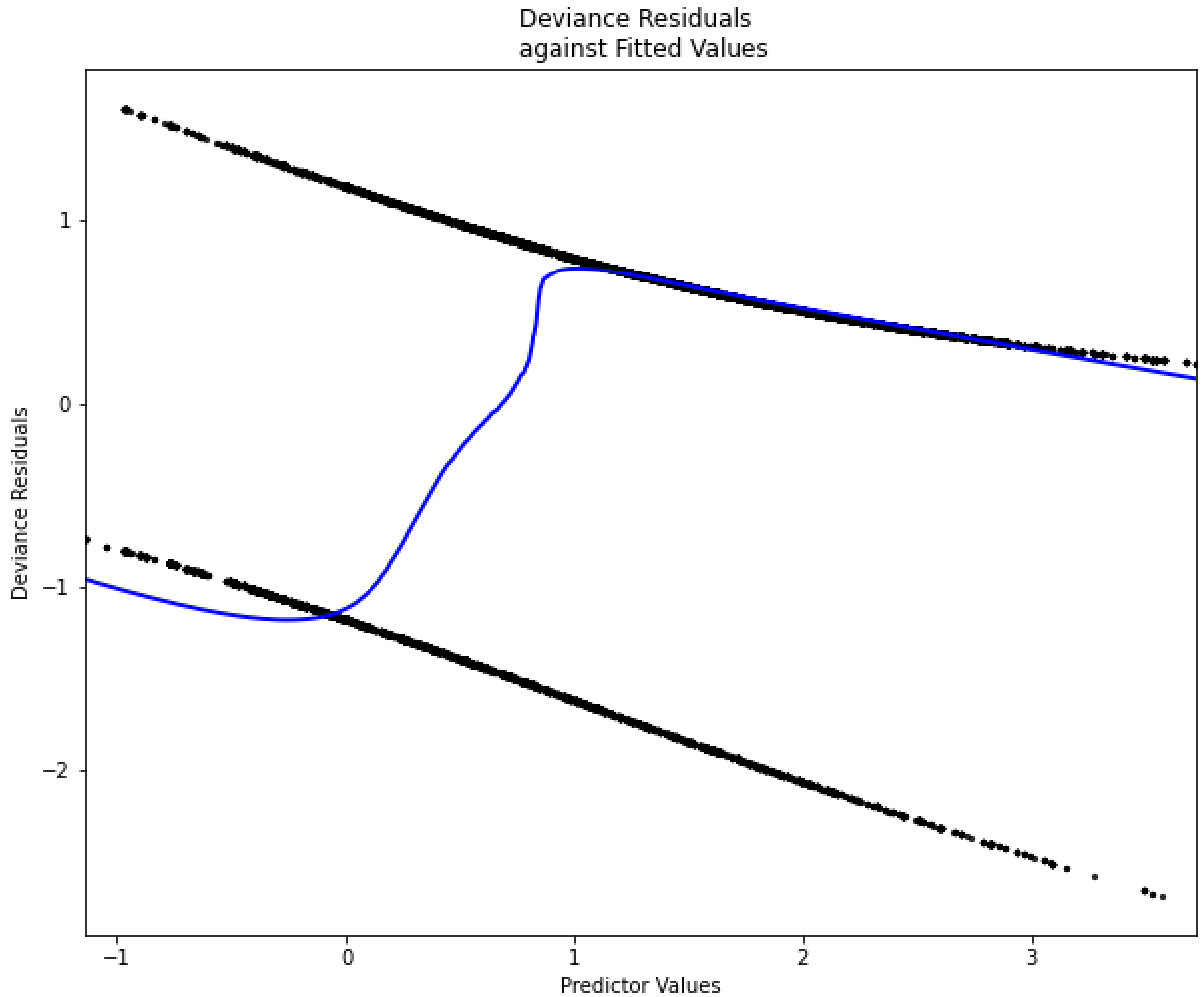
The presence of the independent variables increases or decreases the log odds of the presence of the dependent variable based on the sign of the parameter coefficients. The government regulations of school closures, travel restrictions, state of emergency declarations, wage support, tax credits, and 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. μ̂i 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 [226…

```
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()
```



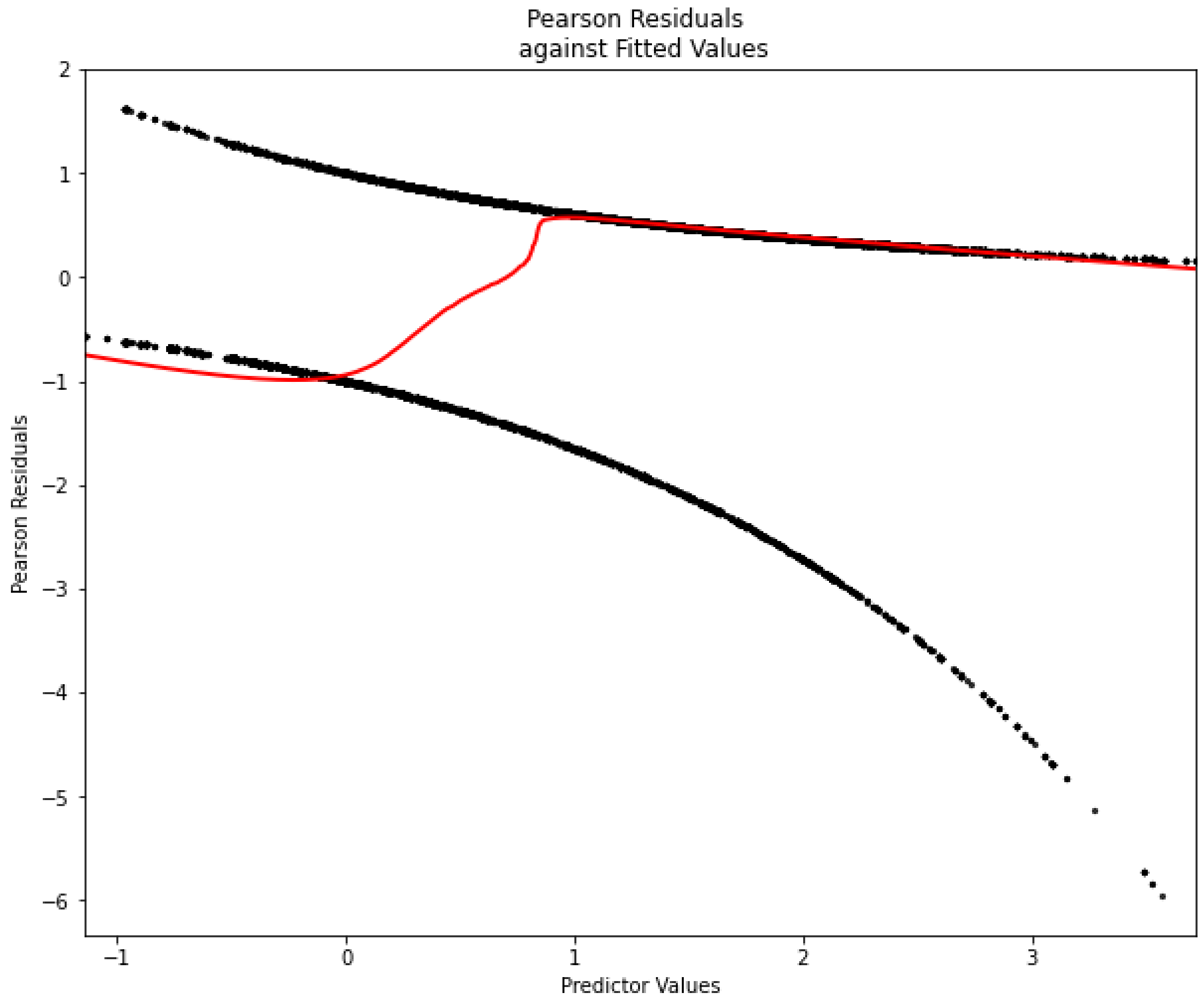
$$PearsonResiduals = \sum_{i}^{N} \frac{y_i - \hat{\mu}_i}{\sqrt{\hat{\mu}_i(n_i - \hat{\mu}_i)/n_i}}$$

In [227…

```
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[227…

```
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.

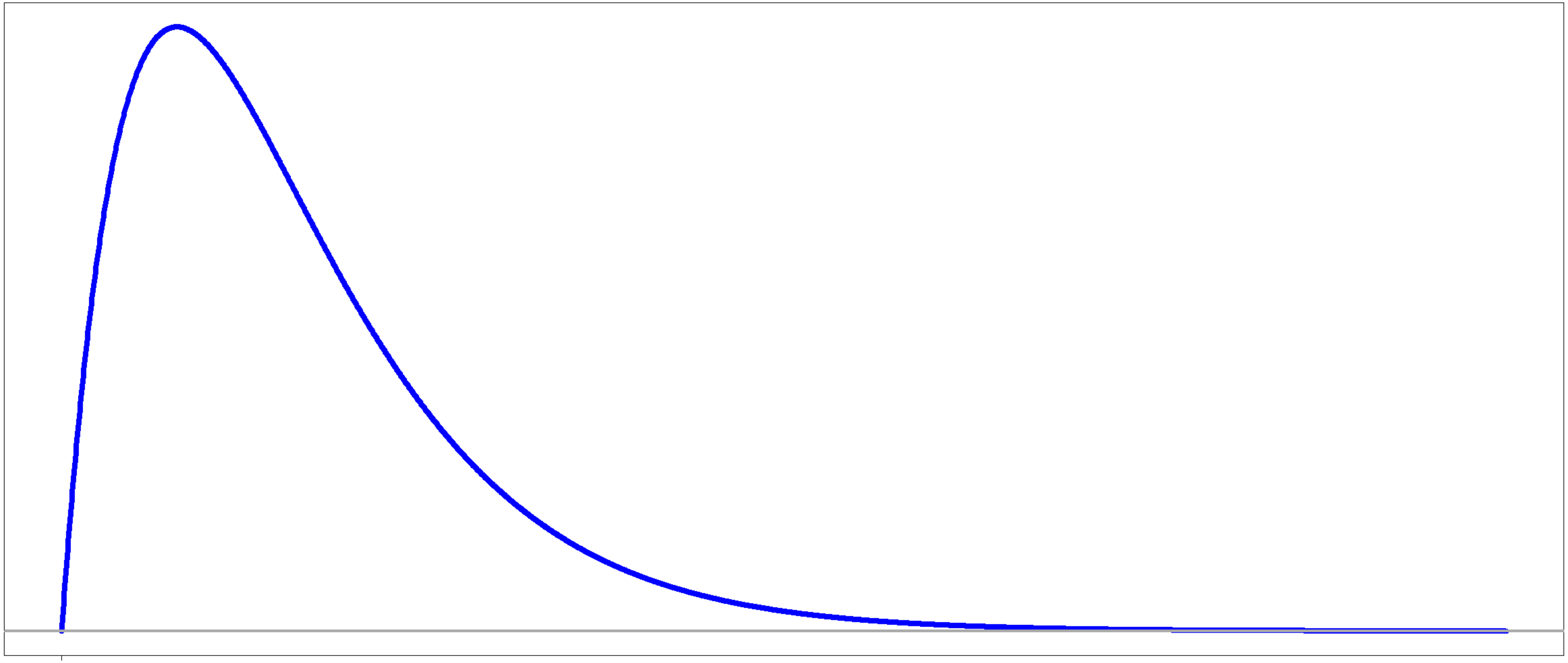
$$\chi 2 = \sum_i^N \frac{(O_i - E_i)^2}{E_i}$$

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

In [503…

```
Image(filename='chi-square.png')
```

Out[503…



$$H0 : \mu O = \mu E$$

$$H1 : \mu O \neq \mu E$$

In [228…
```
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: -153841.8834477555 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 [233…
```
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 [229…

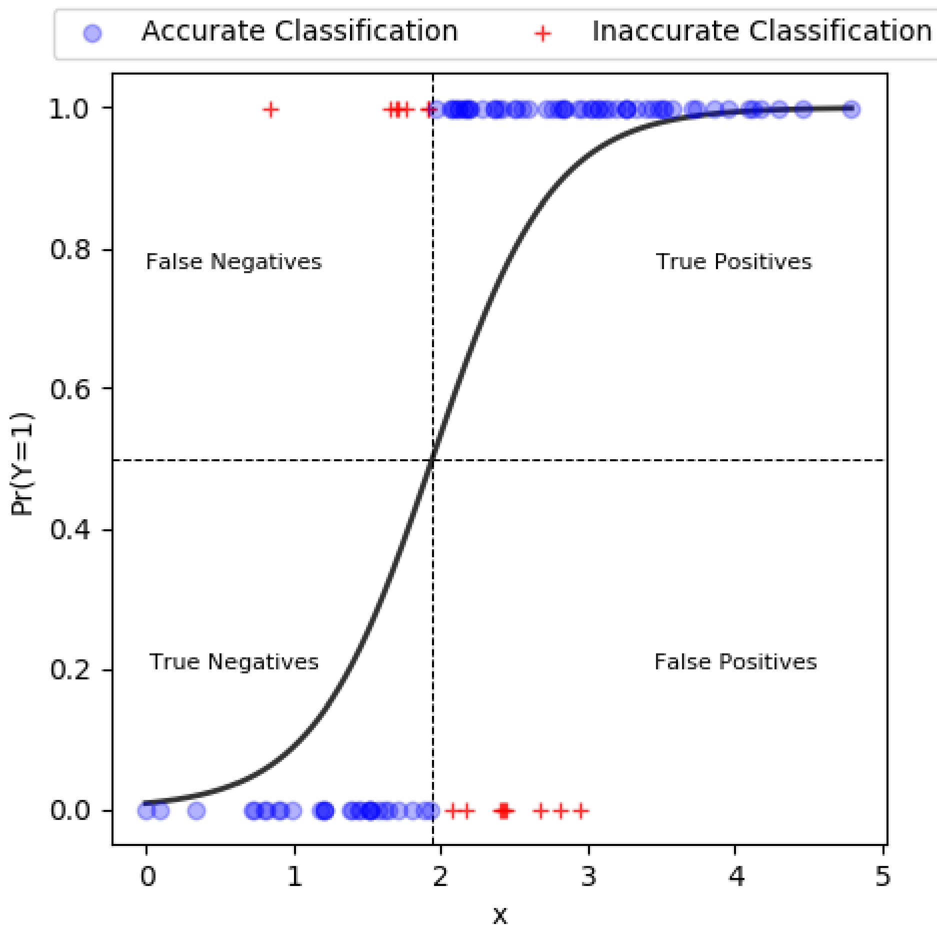
```
for i,l in zip(logistic_regression.coef_[0][1:],X_train.columns):
    odds_difference=i-logistic_regression.coef_[0][0]
    odds_ratio=np.exp(odds_difference)
    print(f'{l}: {odds_ratio}')
```

```
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]:

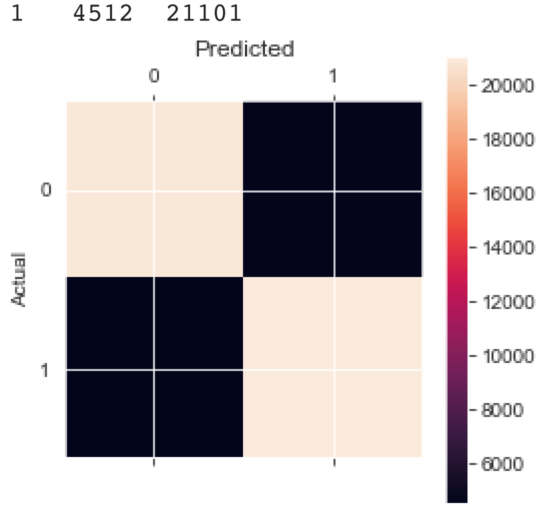


For the train set, the model has 20765 true positives, 20242 true negatives, 4834 false positives, and 4319 false negatives.

In [240…

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

```
Confusion Matrix
----------------
        0       1
0   20951    4688
1    4512   21101
```



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

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

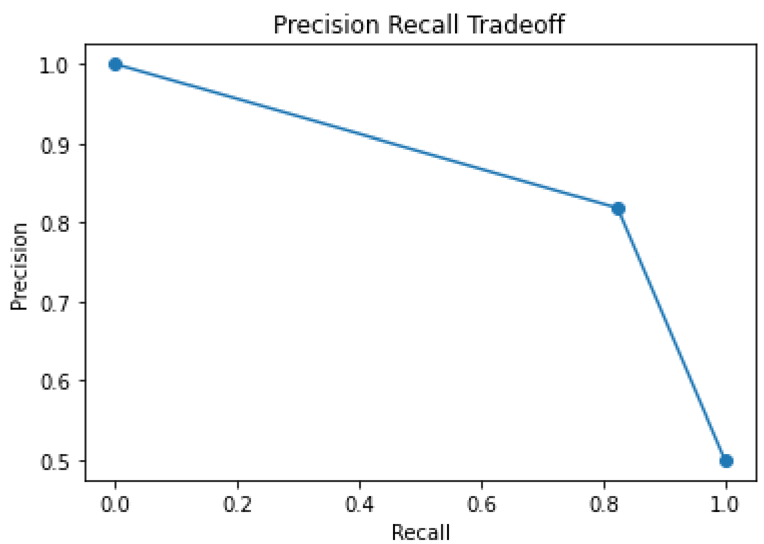
$$f1 = \frac{2 * precision * recall}{precision + recall}$$

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

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

In [235…
```
Metrics(y_train, y_hat_train)
```

```
Precision Score: 0.8182170692931094
Recall Score: 0.8238394565259829
F1 Score: 0.8210186374071048
Accuracy Score: 0.8204948099586358
Specificity Score: 0.8228017122884185
```



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

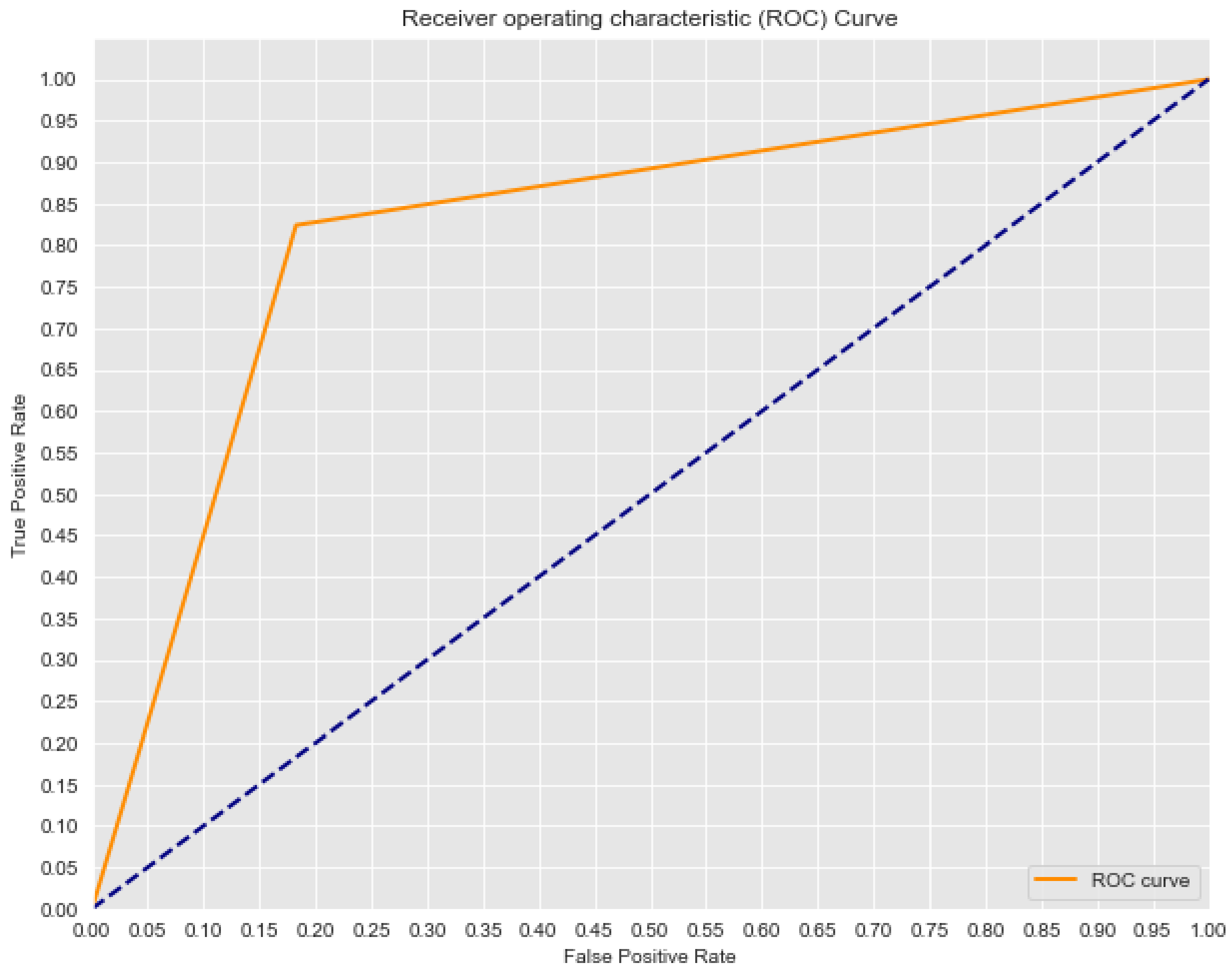
$$FalsePositiveRate = \frac{FP}{FP+TN}$$

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

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

In [236…
```
roc(y_train, y_hat_train)
```

```
AUC: 0.820496505828419
```

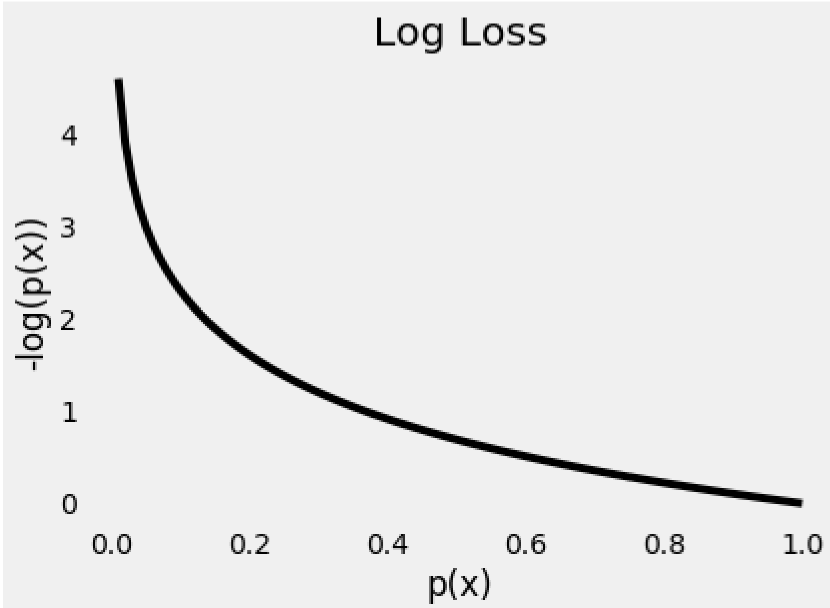


The log loss, a cost function, is the cross entropy between the distribution of the true labels and the predictions. Entropy measures unpredictability and cross entropy incorporates the entropy of the predicted distribution with of the true distribution. The log loss multiplies -1 by the log likelihood 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]:

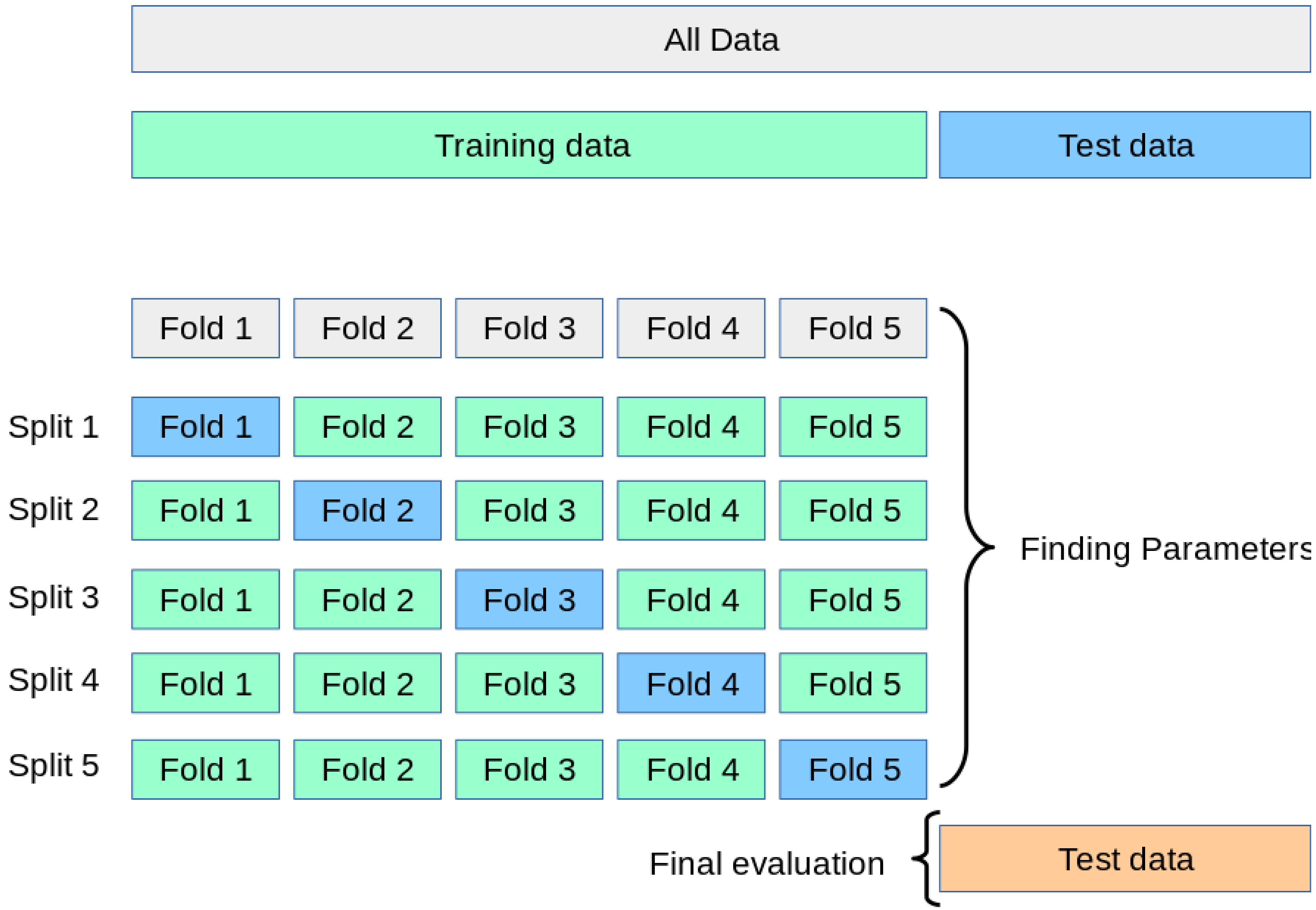


Cross validation trains models on k fold subsets of data and evaluates the models on the complementary subset of the data.

In [357…

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

Out[357…



In [237…
```
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: -0.4313802602717627
```

Test model on test set.

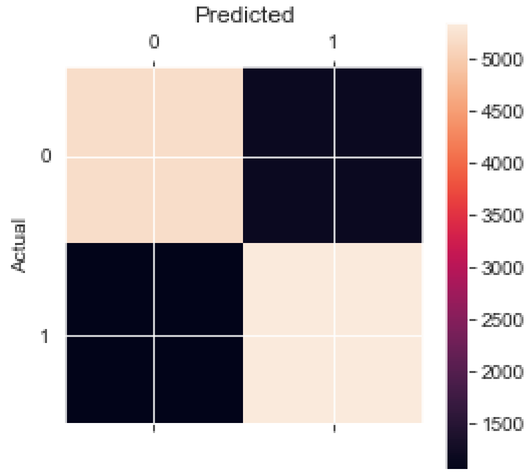
In [238…
```
logistic_regression.fit(X_test, y_test)

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 [239…
```
con_mat(y_test, y_hat_test)
```

```
Confusion Matrix
----------------
      0     1
```

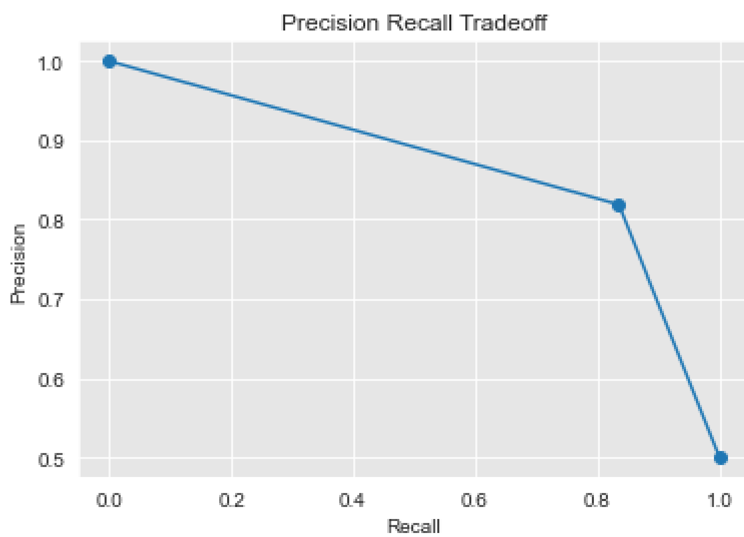
```
0  5207  1187
1  1058  5362
```



In [241…

```
Metrics(y_test, y_hat_test)
```

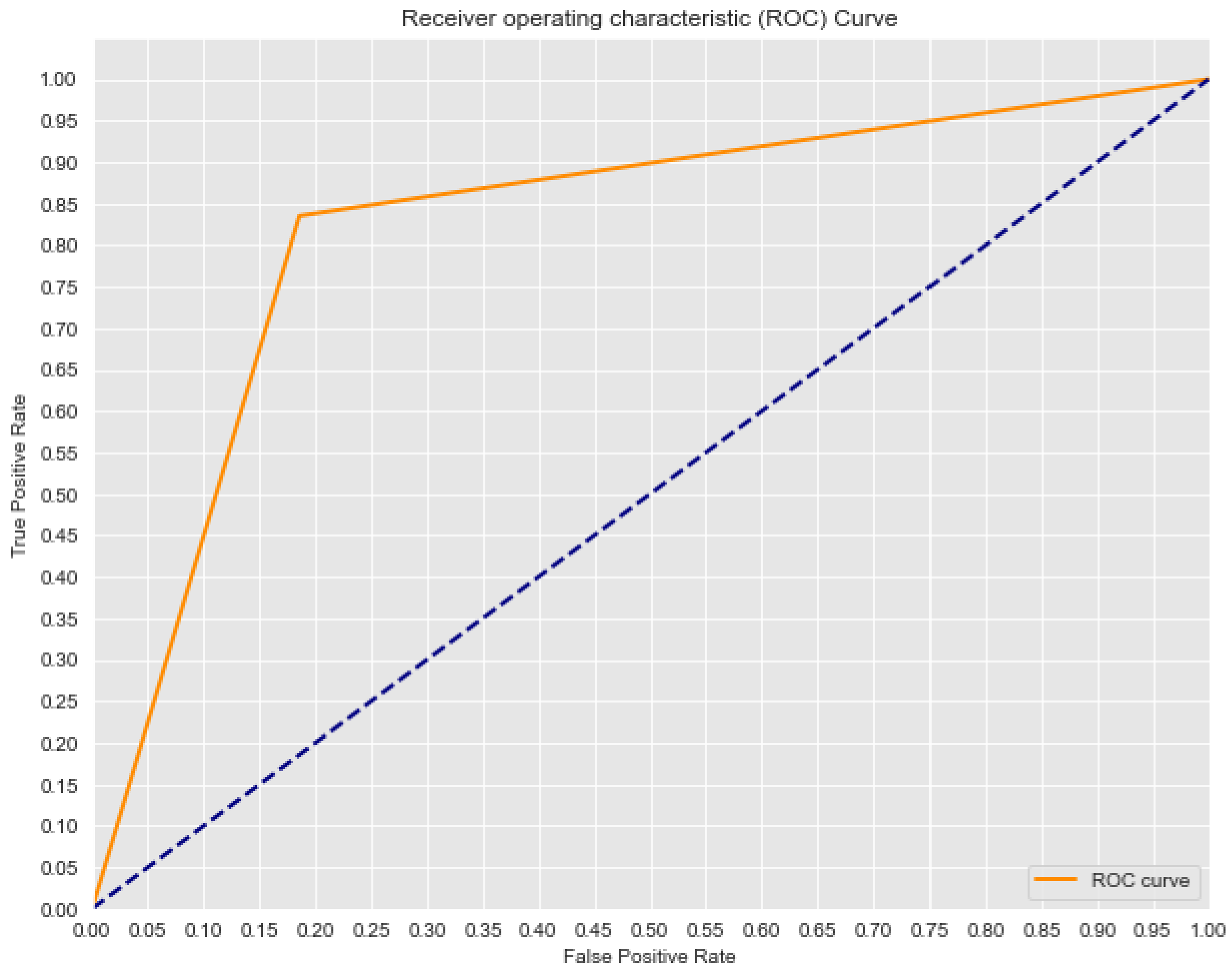
```
Precision Score: 0.8187509543441747
Recall Score: 0.835202492211838
F1 Score: 0.8268949032307811
Accuracy Score: 0.824800998907445
Specificity Score: 0.8311252992817239
```



In [242…

```
roc(y_test,y_hat_test)
```

```
AUC: 0.8247798510480522
```



Next is feature engineering for XGBoost model.

Feature Engineering

Encode cases_xgb to three numbers.

In [243…
```
cases_xgb=cases_xgb.astype('category')
cases_xgb=cases_xgb.cat.codes
```

In [245…
```
y=cases_xgb
X=df.drop(['cases'], axis=1)
```

In [246…
```
X_resampled, y_resampled = oversample.fit_resample(X, y)
```

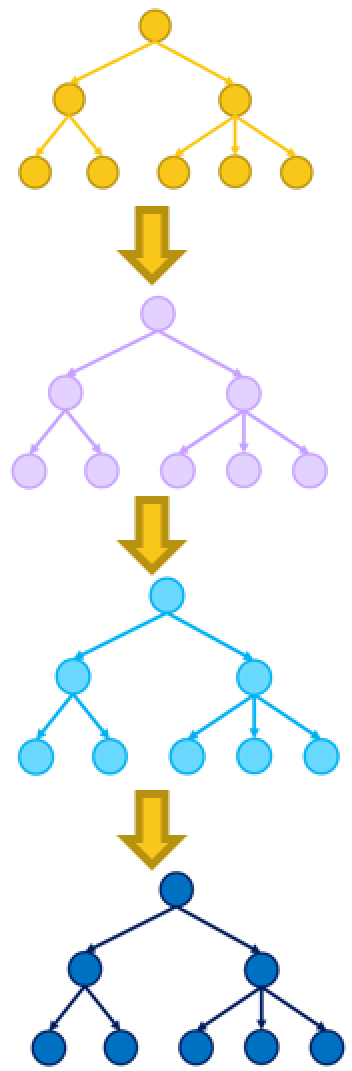
In [247…
```
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

XGBoost Model

XGBoost is a type of tree ensemble model, which contains a set of classification or regression trees (CART) that have a score for each leaf and build a new tree optimized upon the error of the previous tree leaves.

In [474…
```
Image(filename='xgb.png')
```

Out[474…



K is the number of trees and f is the CART function.

$$\hat{y}_i = \sum_k^K f_k(x_i)$$

The objective function to be optimized is the following where l is the loss function and Ω is the regularization penalty term.

$$\zeta = \sum_i^N l(y_i, \hat{y}_i) + \sum_k^K \Omega(f_k)$$

Use an additive strategy to minimize ζ by adding one new tree at a time represented by step t.

$$\hat{y}_i^{(t)} = \sum_{k^{(t)}}^K f_{k^{(t)}}(x_i) = \hat{y}_i^{(t-1)} + f_{k^{(t)}}(x_i)$$

$$\zeta = \sum_i^N l(y_i, \hat{y}_i^{(t)}) + \sum_{k^{(t)}}^K \Omega(f_{k^{(t)}})$$

Take the Taylor expansion of the loss function up to the second order.

$$g_i = \frac{\partial l(y_i, \hat{y}_i^t)}{\partial \hat{y}_i^t}$$

$$h_i = \frac{\partial^2 l(y_i, \hat{y}_i^t)}{\partial \hat{y}_i^t}$$

$$\zeta = \sum_i^N [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_{k^{(t)}}(x_i) + \frac{1}{2} h_i f_{k^{(t)}}^2(x_i)] + \sum_{k^{(t)}}^K \Omega(f_{k^{(t)}})$$

w is the vector of leaf scores, and q is a function assigning each data point to the corresponding leaf, and T is the number of leaves.

$$f_{k^{(t)}}(x_i) = w_{q_t(x)} \in \mathbb{R}^T$$

The regularization term, Ω, is defined as the following where w is the sum of residuals divided by the number of residuals plus lambda.

$$w_j = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

$$\Omega(f_{k^{(t)}}) = \gamma T + \frac{1}{2} \lambda \sum_j^T w_j^2$$

In [468…
```
xgb = XGBClassifier()
xgb.fit(X_train, y_train)
```

Out[468…
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1,
              objective='multi:softprob', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=None, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [469…
```
y_hat_train_xgb = xgb.predict(X_train)
```

Determine feature importance and statistical significance of independent variables on dependent variable.

In [501…
```
importance_gb = xgb.feature_importances_

for i,c in zip(importance_gb,X_train):
    #add one to the expected values because zero value in denominater causes zero division error.
    stat, p = chisquare(X_train[c],y_train+1)#Null Hypothesis: no significant difference between the observed and the expected values
    alpha=.05
    if p>alpha:
        s='Don\'t reject null of no significant difference between the observed and the expected values.'
    else:
        s='Reject null of no significant difference between the observed and the expected values.'
    print(f'Feature: {c}, Importance: {i}, Statistical Significance: {p}, {s}\n')
```

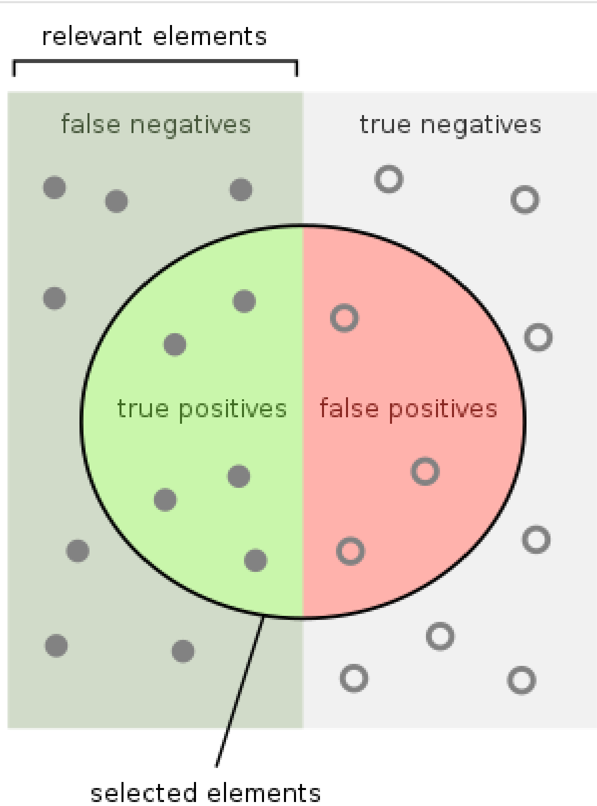
```
Feature: school, Importance: 0.02251388132572174, Statistical Significance: 9.942221787860725e-52, Reject null of no significant difference between the observed and the expected values.

Feature: domestic, Importance: 0.02528320625424385, Statistical Significance: 0.0, Reject null of no significant difference between the observed and the expected values.

Feature: travel, Importance: 0.021120978519320488, Statistical Significance: 1.0, Don't reject null of no significant difference between the observed and the expected values.

Feature: travel_dom, Importance: 0.01877639815211296, Statistical Significance: 0.0, Reject null of no significant difference between the observed and the expected values.
```

```
Feature: curf, Importance: 0.024422796443104744, Statistical Significance: 0.0, Reject null of no significant difference between the observed and the expected values.

Feature: mass, Importance: 0.32087960839271545, Statistical Significance: 1.0, Don't reject null of no significant difference between the observed and the expected values.

Feature: elect, Importance: 0.040125828236341476, Statistical Significance: 0.0, Reject null of no significant difference between the observed and the expected values.

Feature: sport, Importance: 0.08951212465763092, Statistical Significance: 1.0, Don't reject null of no significant difference between the observed and the expected values.

Feature: rest, Importance: 0.026595894247293472, Statistical Significance: 2.0192463404757942e-203, Reject null of no significant difference between the observed and the expected value
s.

Feature: testing, Importance: 0.024204546585679054, Statistical Significance: 0.0, Reject null of no significant difference between the observed and the expected values.

Feature: masks, Importance: 0.043223921209573746, Statistical Significance: 2.4865471896166705e-144, Reject null of no significant difference between the observed and the expected value
s.

Feature: surveillance, Importance: 0.041012052446603775, Statistical Significance: 0.0, Reject null of no significant difference between the observed and the expected values.

Feature: state, Importance: 0.0272047221660614, Statistical Significance: 0.0, Reject null of no significant difference between the observed and the expected values.

Feature: cash, Importance: 0.07483754307031631, Statistical Significance: 1.8051054606625154e-76, Reject null of no significant difference between the observed and the expected values.

Feature: wage, Importance: 0.033472537994384766, Statistical Significance: 5.266513792448481e-99, Reject null of no significant difference between the observed and the expected values.

Feature: credit, Importance: 0.02976750209927559, Statistical Significance: 1.3112475835026576e-133, Reject null of no significant difference between the observed and the expected value
s.

Feature: taxc, Importance: 0.026565508916974068, Statistical Significance: 2.448037347417911e-218, Reject null of no significant difference between the observed and the expected values.

Feature: taxd, Importance: 0.04202548414468765, Statistical Significance: 3.226035480465689e-79, Reject null of no significant difference between the observed and the expected values.

Feature: export, Importance: 0.02952960692346096, Statistical Significance: 0.0, Reject null of no significant difference between the observed and the expected values.

Feature: rate, Importance: 0.038925908505916595, Statistical Significance: 2.3683474955598234e-19, Reject null of no significant difference between the observed and the expected values.
```

In [502…

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

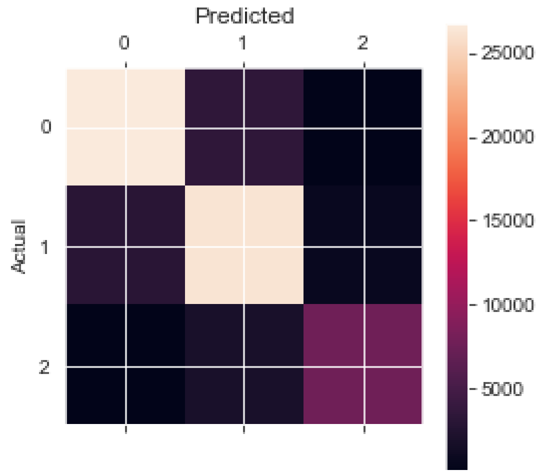
Out[502…



In [471…

```
con_mat(y_train,y_hat_train_xgb)
```

```
Confusion Matrix
----------------
       0      1     2
0  26786   3548   182
1   3191  26358   839
2    219   1961  7845
```



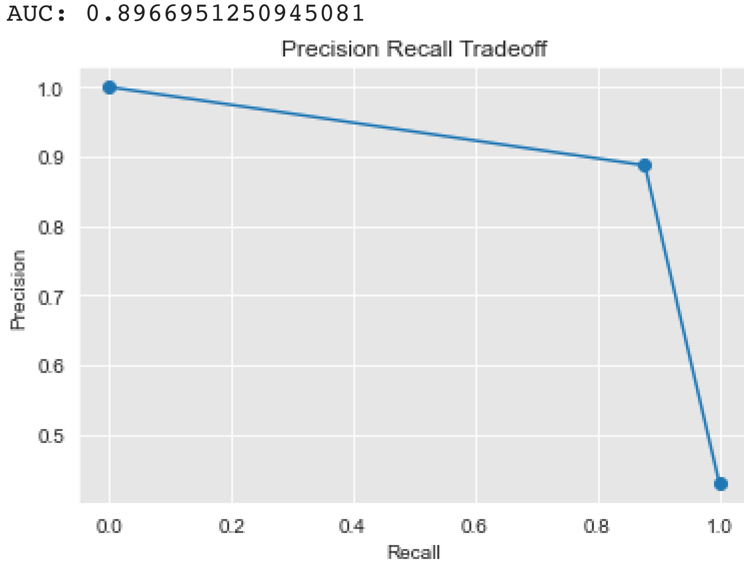
In [472…

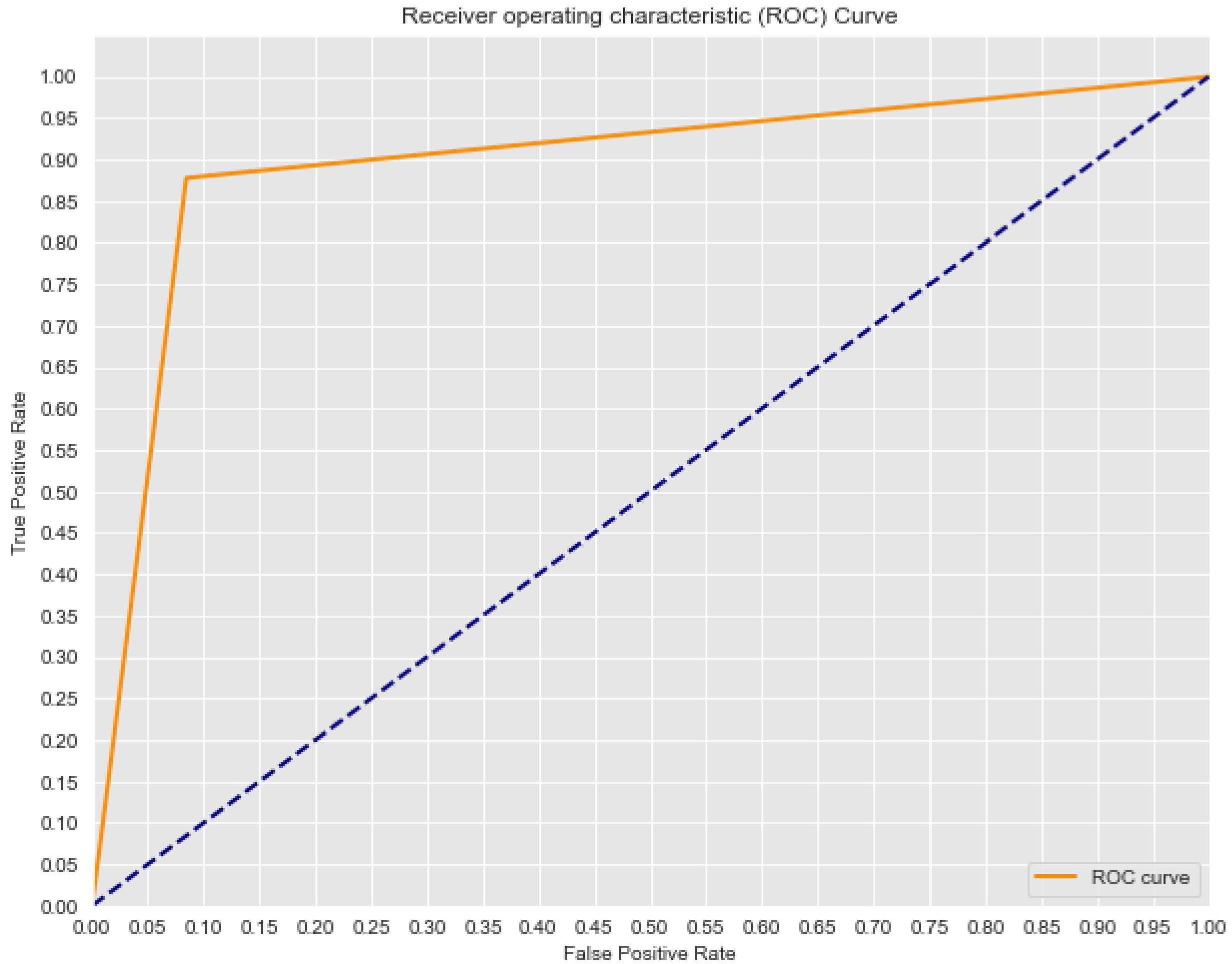
```
actual_class0=y_train==0
actual_class1=y_train==1
actual_class2=y_train==2
predictions_class0=y_hat_train_xgb==0
predictions_class1=y_hat_train_xgb==1
predictions_class2=y_hat_train_xgb==2
a=[actual_class0,actual_class1,actual_class2]
p=[predictions_class0,predictions_class1,predictions_class2]
```

In [473…

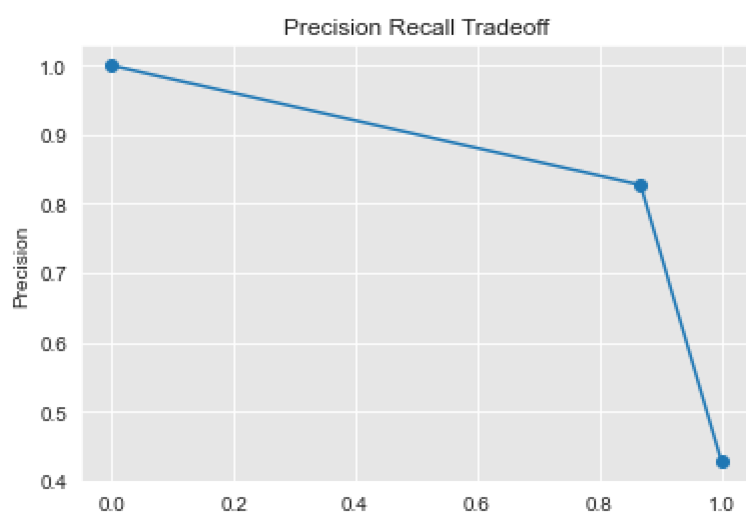
```
multiclass(xgb,X_train,a,p,y_train,'train')
```

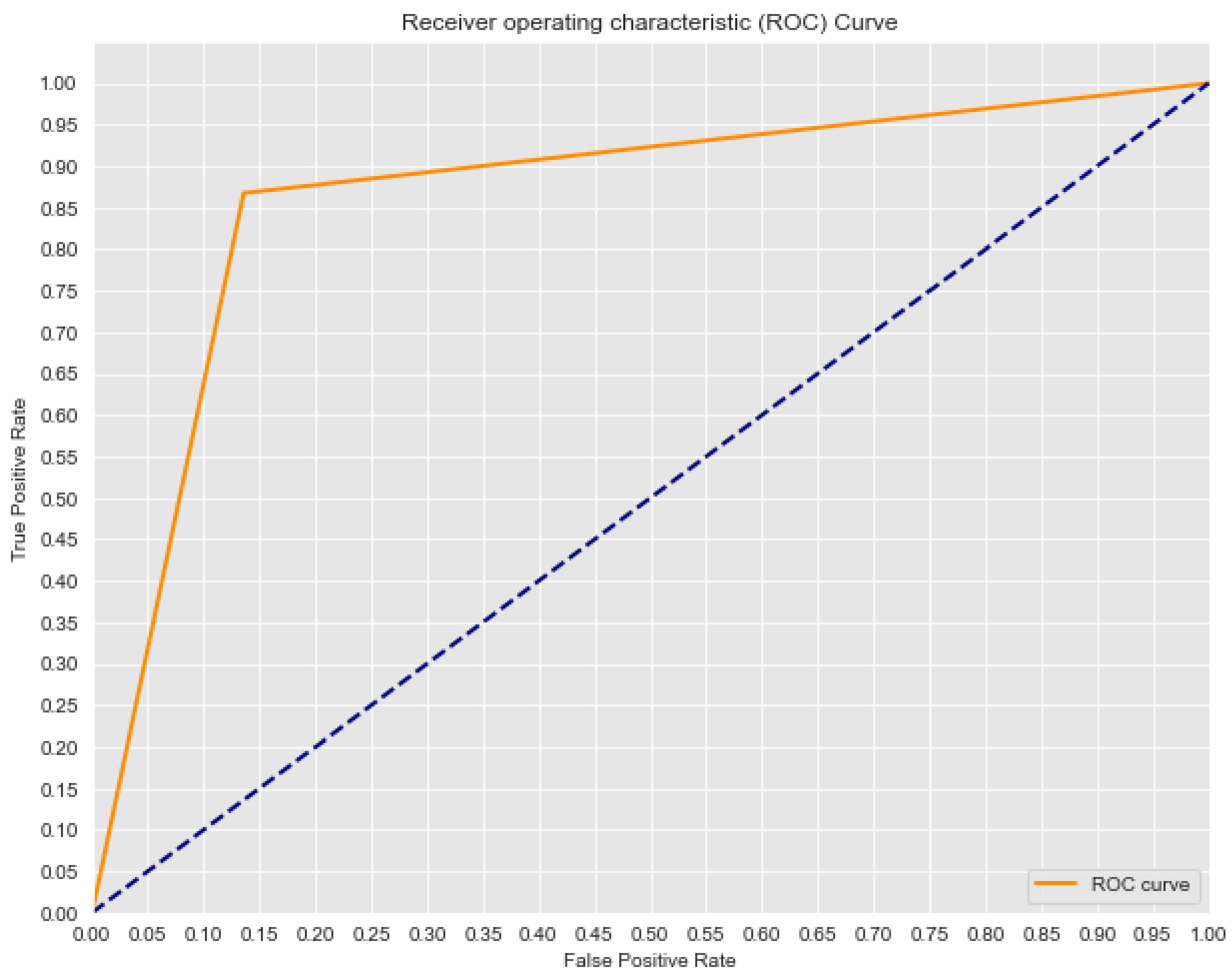
```
Class:0
Precision Score: 0.8870711352497019
Recall Score: 0.8777690391925548
F1 Score: 0.8823955725392015
Accuracy Score: 0.8993359556739838
Specificity Score: 0.908428055876071
AUC: 0.8966951250945081
```



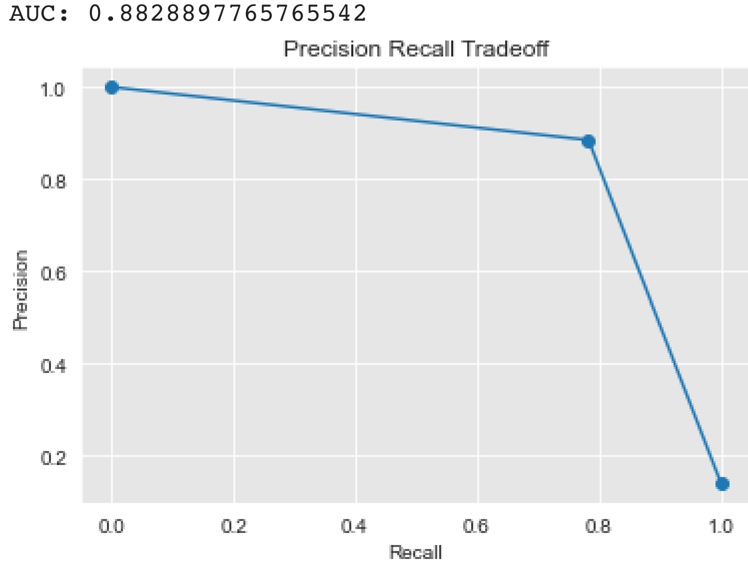


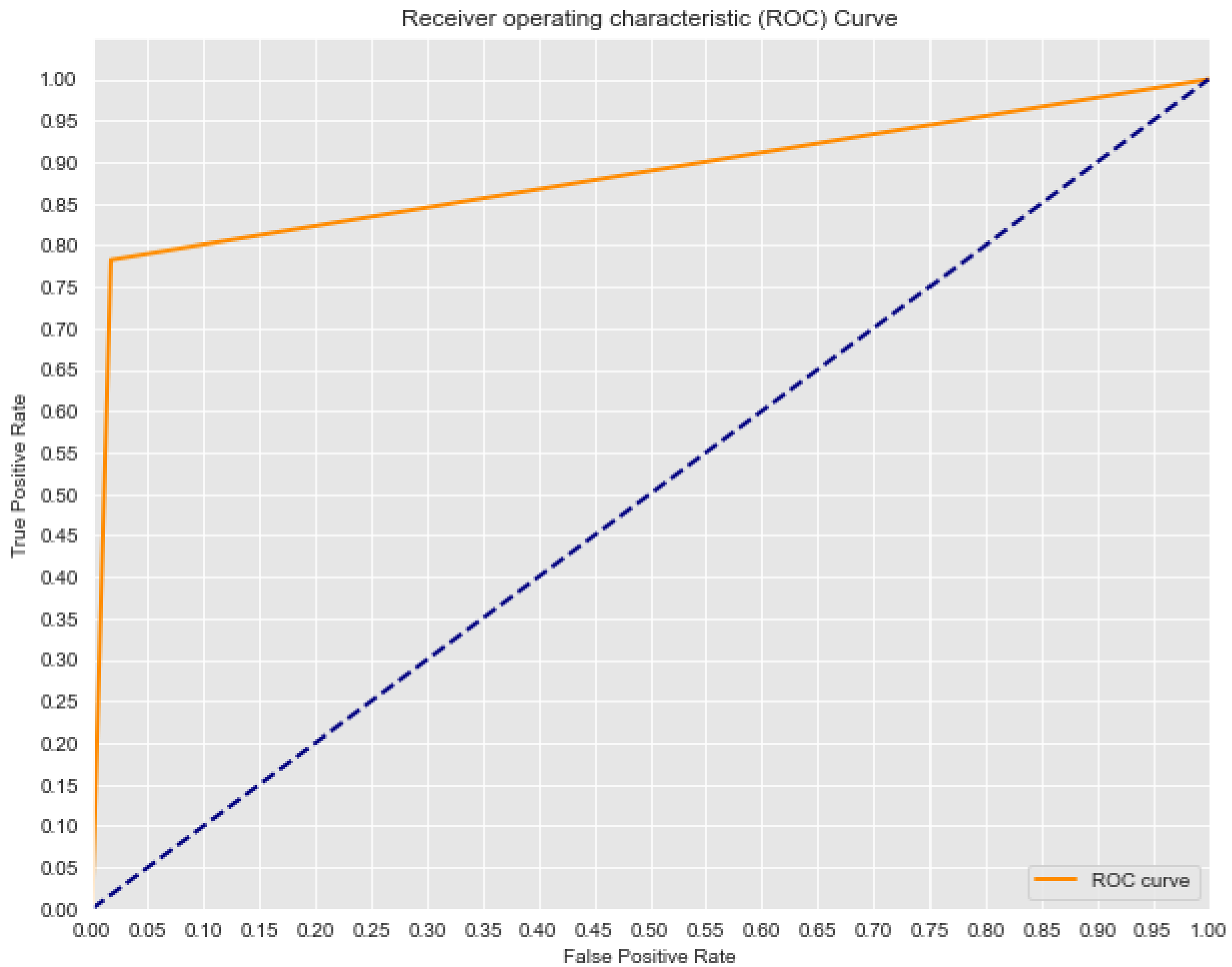

```
Class:1
Precision Score: 0.8271252392757398
Recall Score: 0.8673818612610241
F1 Score: 0.8467753594088827
Accuracy Score: 0.8655134007246683
Specificity Score: 0.8968306794326968
AUC: 0.8657473673242296
```






```
Class:2
Precision Score: 0.8848409654861268
Recall Score: 0.7825436408977556
F1 Score: 0.8305542321740511
Accuracy Score: 0.9548703633210676
Specificity Score: 0.964874401817508
AUC: 0.8828897765765542
```






```
_______________________________________________________________________
Class 0 Cross Validated ROC AUC Score: 0.8937466774747875
Class 1 Cross Validated ROC AUC Score: 0.8612240370381536
Class 2 Cross Validated ROC AUC Score: 0.8768997289082318
```

Test the model.

In [476…

```
xgb.fit(X_test, y_test)
```

Out[476…

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1,
              objective='multi:softprob', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=None, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

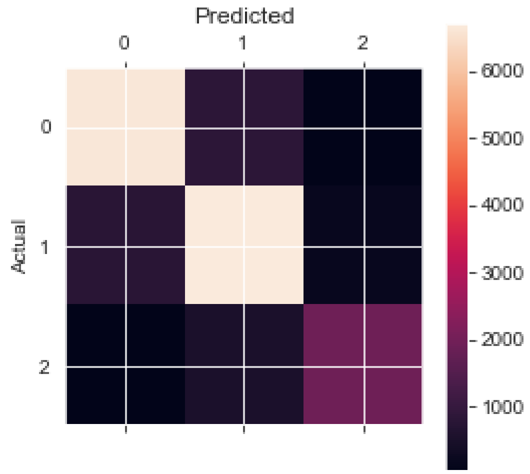
In [477…

```
y_hat_test_xgb = xgb.predict(X_test)
```

In [478…

```
con_mat(y_test,y_hat_test_xgb)
```

```
Confusion Matrix
----------------
      0     1     2
0  6658   843    49
1   780  6718   180
2    62   518  1925
```



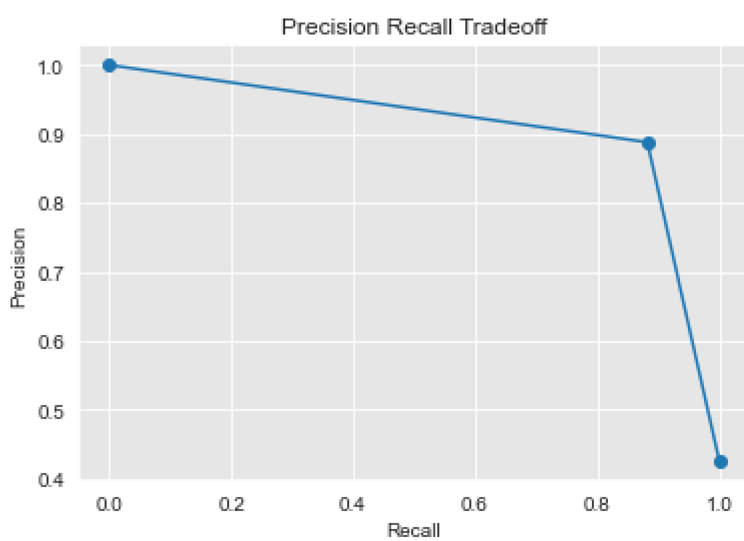
In [479…

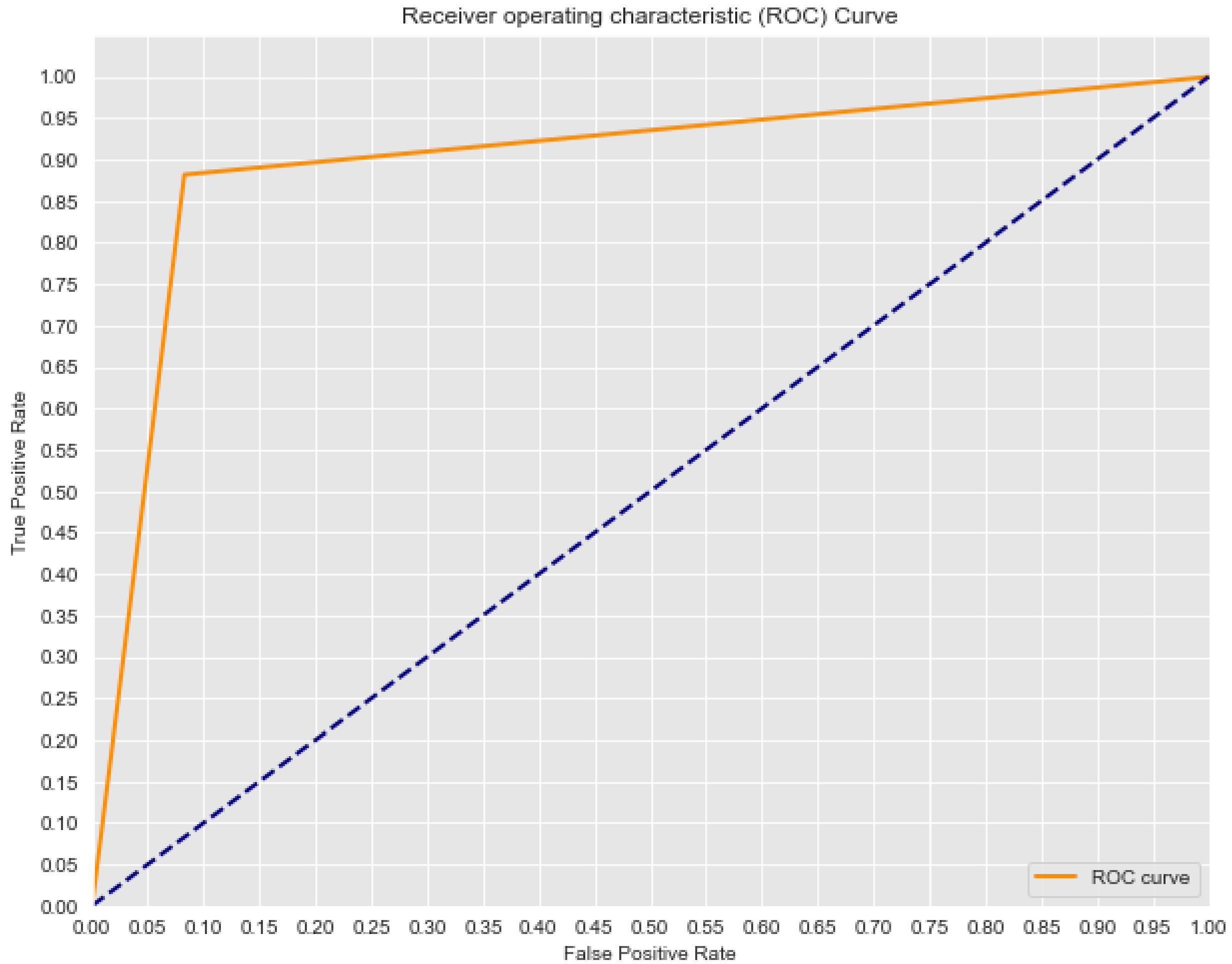
```
actual_class0=y_test==0
actual_class1=y_test==1
actual_class2=y_test==2
predictions_class0=y_hat_test_xgb==0
predictions_class1=y_hat_test_xgb==1
predictions_class2=y_hat_test_xgb==2
a=[actual_class0,actual_class1,actual_class2]
p=[predictions_class0,predictions_class1,predictions_class2]
```

In [480…

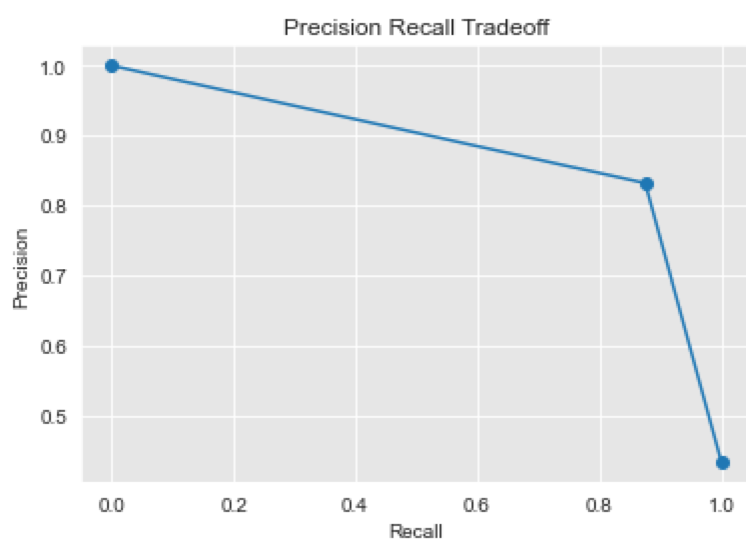
```
multiclass(xgb,X_test,a,p,y_test,'test')
```

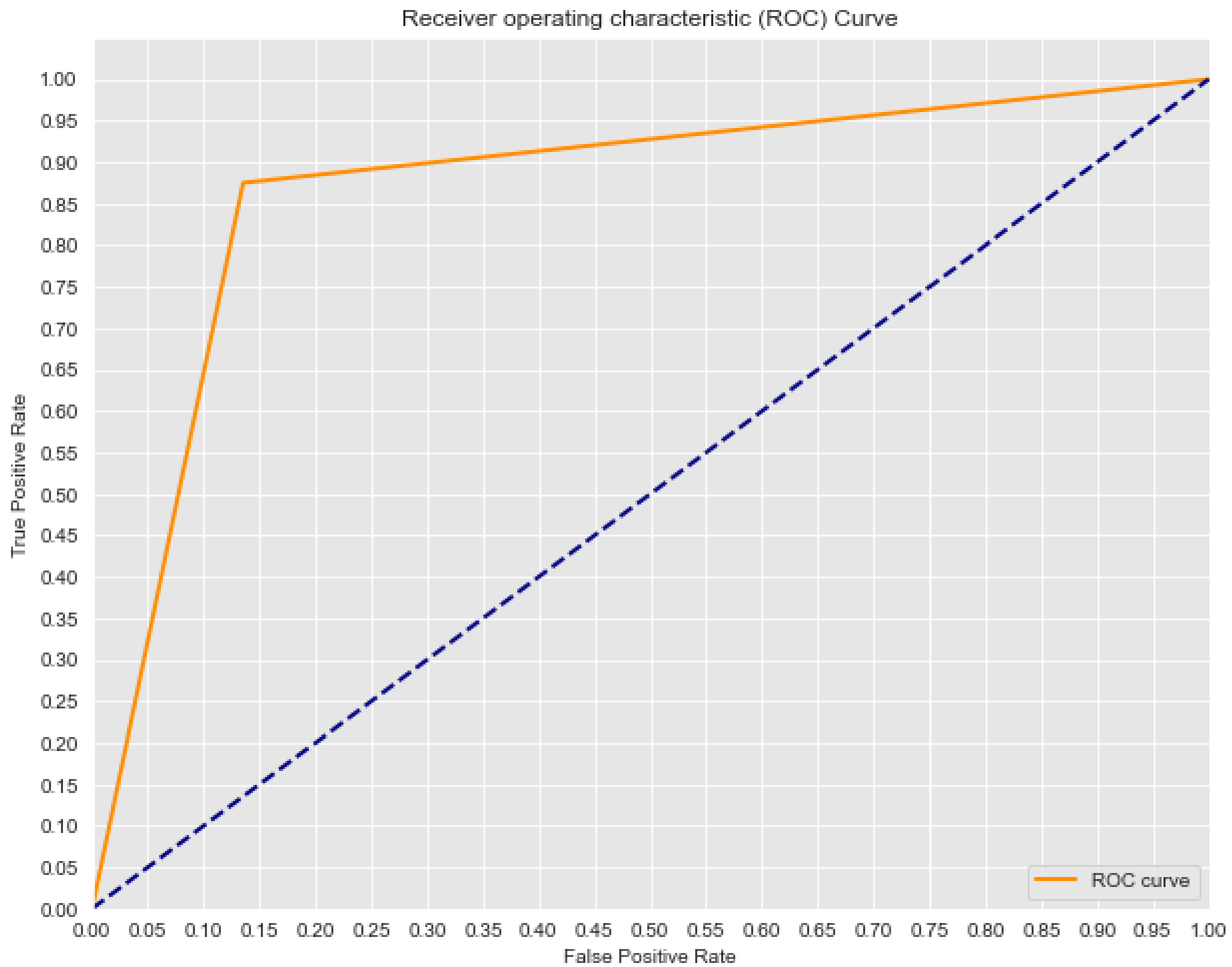
```
Class:0
Precision Score: 0.8877333333333334
Recall Score: 0.8818543046357616
F1 Score: 0.8847840531561463
Accuracy Score: 0.9022162070715615
Specificity Score: 0.912831036841591
AUC: 0.8995837368214652
```



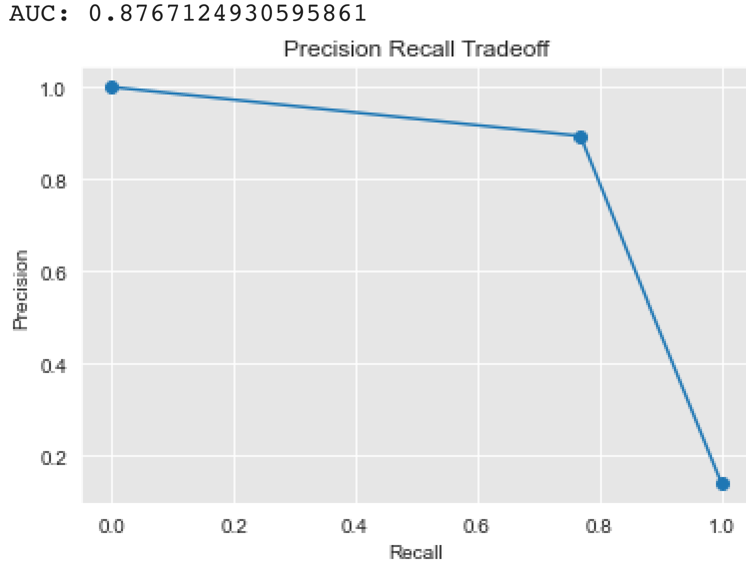


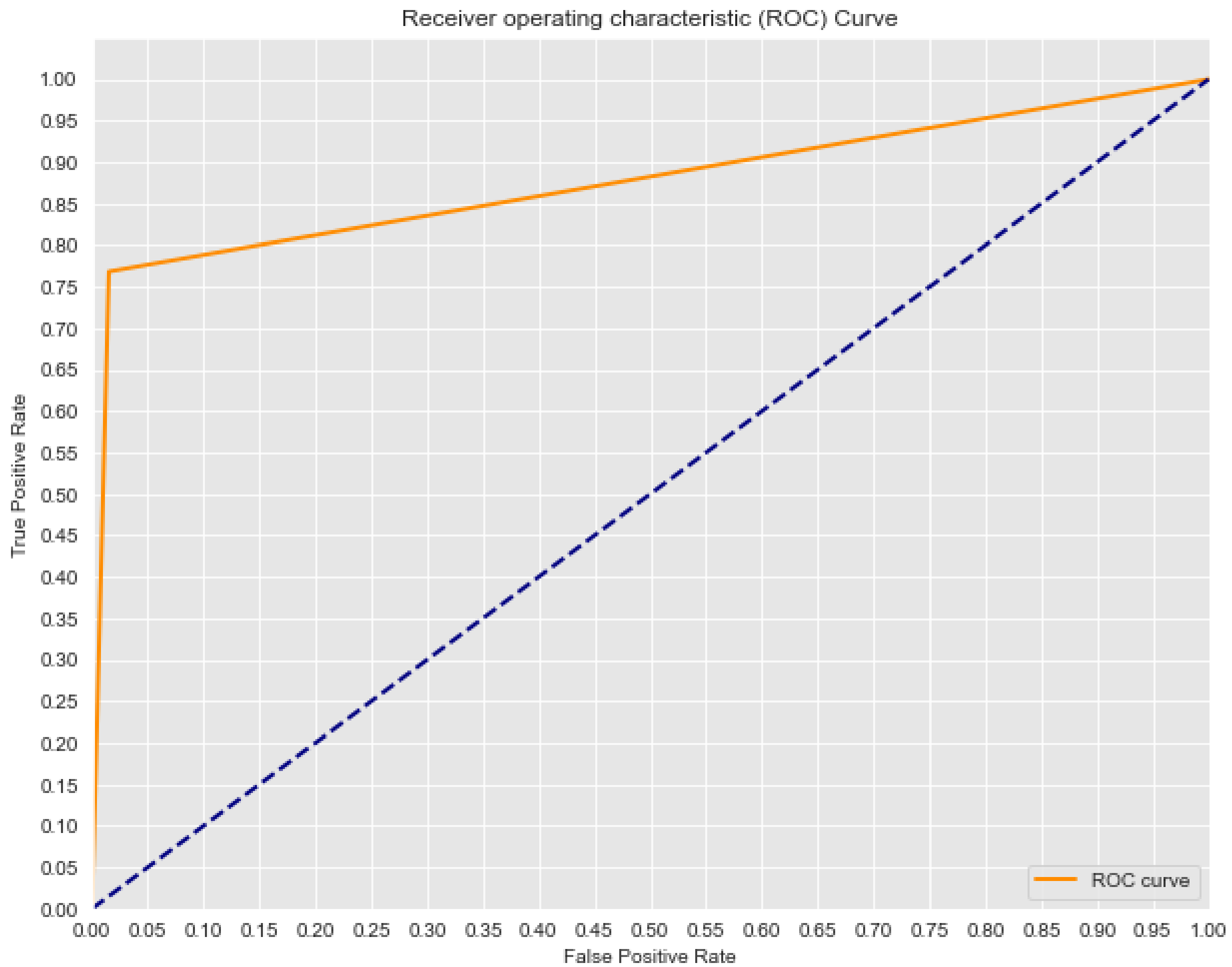

```
Class:1
Precision Score: 0.8315385567520732
Recall Score: 0.8749674394373534
F1 Score: 0.8527003871295297
Accuracy Score: 0.8691140810917498
Specificity Score: 0.9005593536357986
AUC: 0.8698059474660661
```






```
________________________________________________
Class:2
Precision Score: 0.893686165273909
Recall Score: 0.7684630738522954
F1 Score: 0.8263575874651211
Accuracy Score: 0.9543788417075509
Specificity Score: 0.9627703960459593
AUC: 0.8767124930595861
```





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 XGBoost model suggests that no statistically significant feature had significantly greater importance over the others in predicting cases of the virus. 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 where people gather would be sufficient in preventing the spread of SARS-CoV-2.

In [ ]: