Can Machine Learning Predict 4D Winning Numbers?

Prepared by: Cheryl Lim

Agenda

- Introduction
- Methodology
- Process Workflow
- Results
- Conclusions

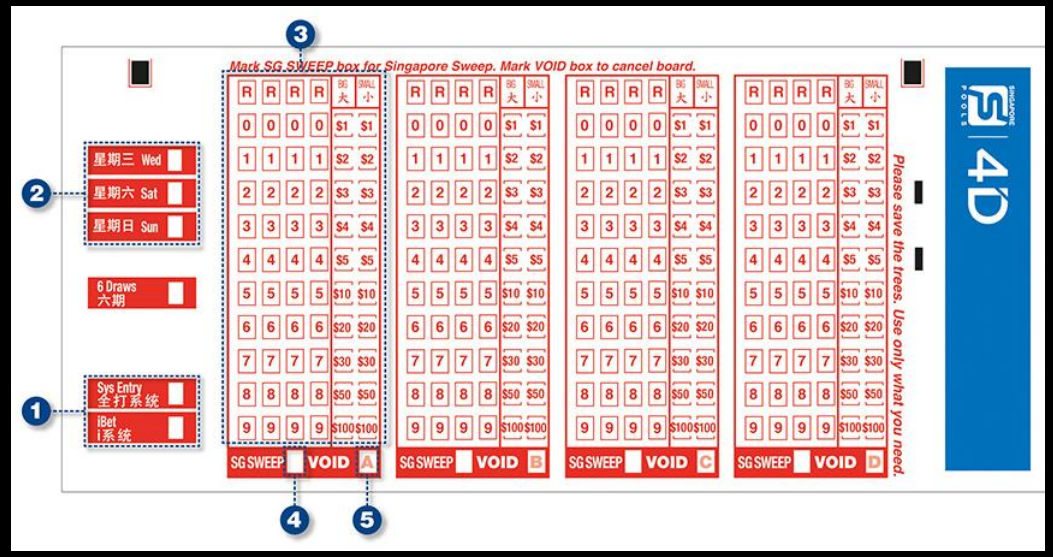
Introduction

SOURCE: SINGAPORE POOLS

How to play 4D? (1)

SOURCE: SINGAPORE POOLS

- Select a four-digit number from 0000 to 9999
- Minimum cost is $1
- Draws take place every Wednesday, Saturday and Sunday at 6.30pm
- Each draw has 23 Prizes:
  - First Prize
  - Second Prize
  - Third Prize
  - 10 Starter Prizes
  - 10 Consolation Prizes



How to play 4D? (2)

SOURCE: SINGAPORE POOLS

Ordinary Entry

1 combination

$1

4D Roll

10 combinations

$10

Ordinary Entry ticket

1. Bet Type
2. Self Pick selection and bet amount
3. Quick Pick selection and bet amount
4. Total bet amount
5. Draw date and number

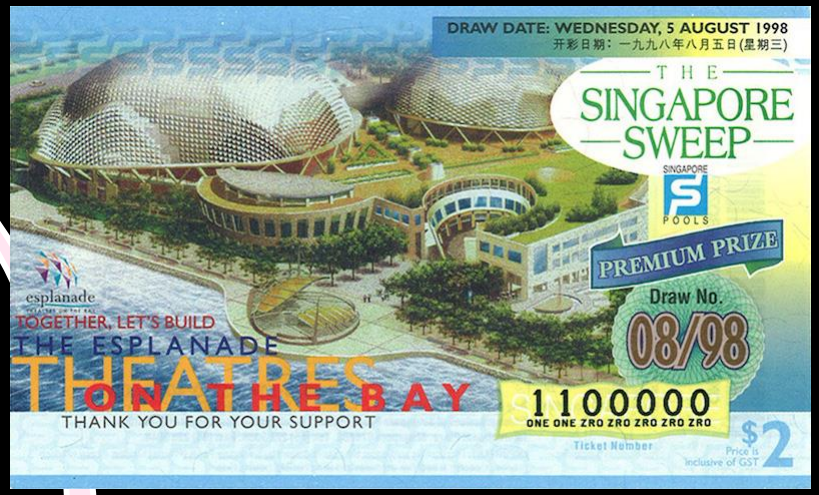
4D Roll ticket

1. 'R' represents any number from 0 to 9
2. Total bet amount
3. Draw date and number

Questions

- Can we predict 4D results?
- Can we predict the number on a given day?
- Can we predict which prize category the number will be?

Methodology



SOURCE: SINGAPORE POOLS

Data Collection (1)

Web Page: Singapore Pools
Check Past Winning Numbers

Tool: Web Scraping using BeautifulSoup and Selenium in Python

Data: Number, Day and Prize

Data Collection (2)

In [1]:
```
!pip install lxml
from bs4 import BeautifulSoup

from selenium import webdriver
import time

import pandas as pd
```

```
Requirement already satisfied: lxml in c:\users\user\anaconda3\lib\site-packages (4.6.1)
```

In [2]:
```
df = pd.DataFrame()

for i in range(0, 500):
    url = "https://www.singaporepools.com.sg/en/product/Pages/4d_cpwn.aspx"

    PATH = 'C:\Program Files (x86)\chromedriver.exe'
    driver = webdriver.Chrome(PATH)

    driver.get(url)

    search = driver.find_element_by_class_name("form-control.text-center.four-d-user-number")
    search.send_keys(i)

    showButton = driver.find_element_by_class_name("btn.btn-orange.form-control.btnShowByDate")
    showButton.click()

    time.sleep(1.5)
    page_source = driver.page_source

    driver.quit()

    soup = BeautifulSoup(page_source, 'lxml')

    table = soup.find("div", {"class": "results-prize-group"})

    for tag in table.find_all("tr")[1:]:
        number = format(i, '04d')

        cells = tag.find_all("td")
        day = cells[0].text
        prize = cells[1].text

        df = df.append({'number': number, 'day': day, 'prize': prize}, ignore_index=True)
```

In [19]:
```
df.to_csv('webscraping1.csv', index=False)
```

Step 1: Load in the required data

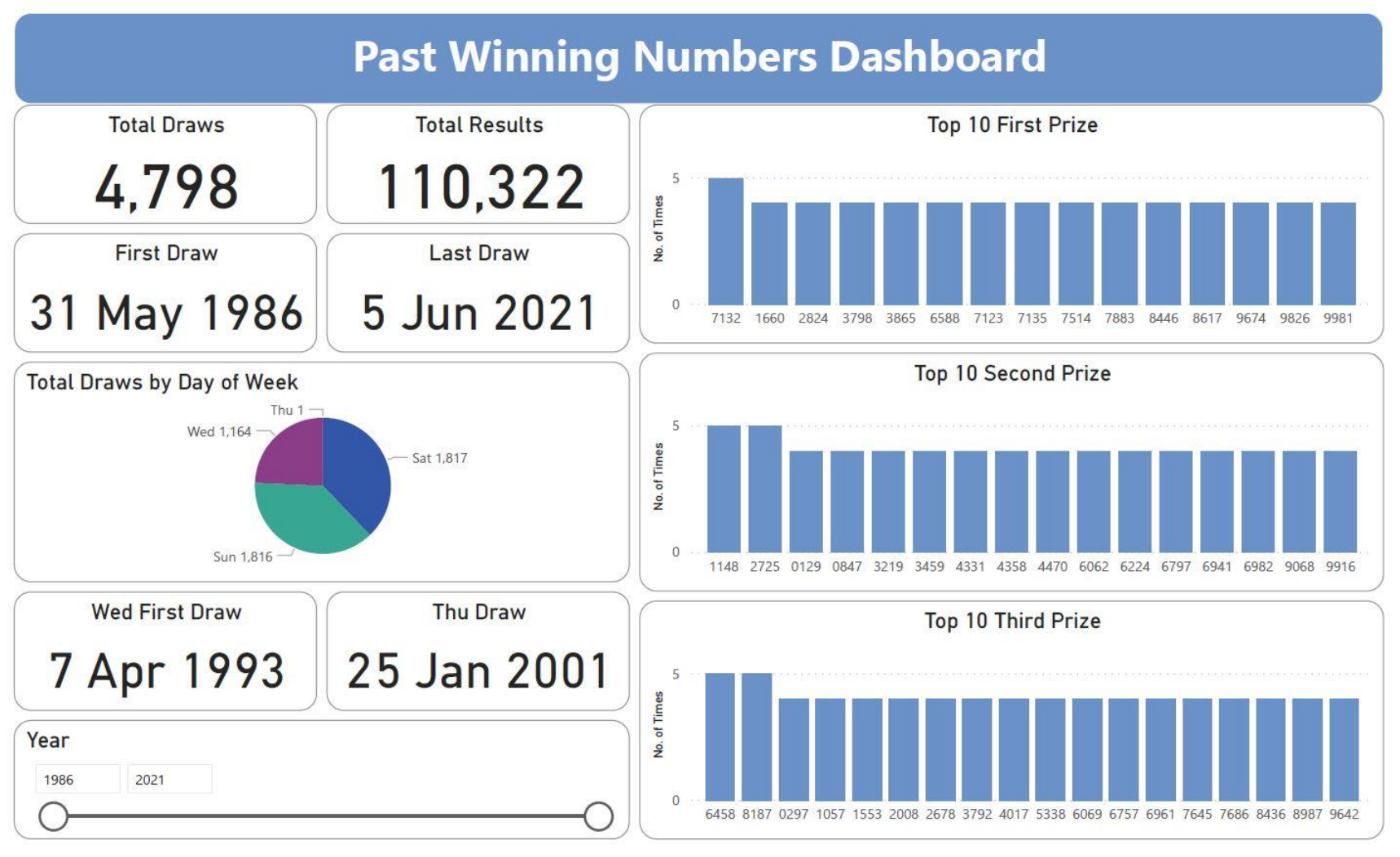
In [2]:
```
# Concatenate 3 CSV files from web scraping

df = pd.concat(
    map(pd.read_csv, ['webscraping1.csv', 'webscraping2.csv', 'webscraping3.csv']), ignore_index=True)
df.head(10)
```

Out[2]:

	day	number	prize
0	Wed, 24 Jun 2020	0	Consolation Prize
1	Sun, 21 Feb 2016	0	Consolation Prize
2	Wed, 20 Jan 2016	0	Consolation Prize
3	Sun, 05 Oct 2014	0	First Prize
4	Sat, 06 Sep 2014	0	Second Prize
5	Sat, 26 Oct 2013	0	Starter Prize
6	Sun, 29 Jan 2012	0	Starter Prize
7	Sat, 06 Aug 2011	0	Consolation Prize
8	Sat, 13 Jun 2009	0	Consolation Prize
9	Sun, 14 May 2006	0	Starter Prize

Dashboard in Power BI (1)



Prize Amount

(for every $1 Small)

1st Prize: $3,000

2nd Prize: $2,000

3rd Prize: $800

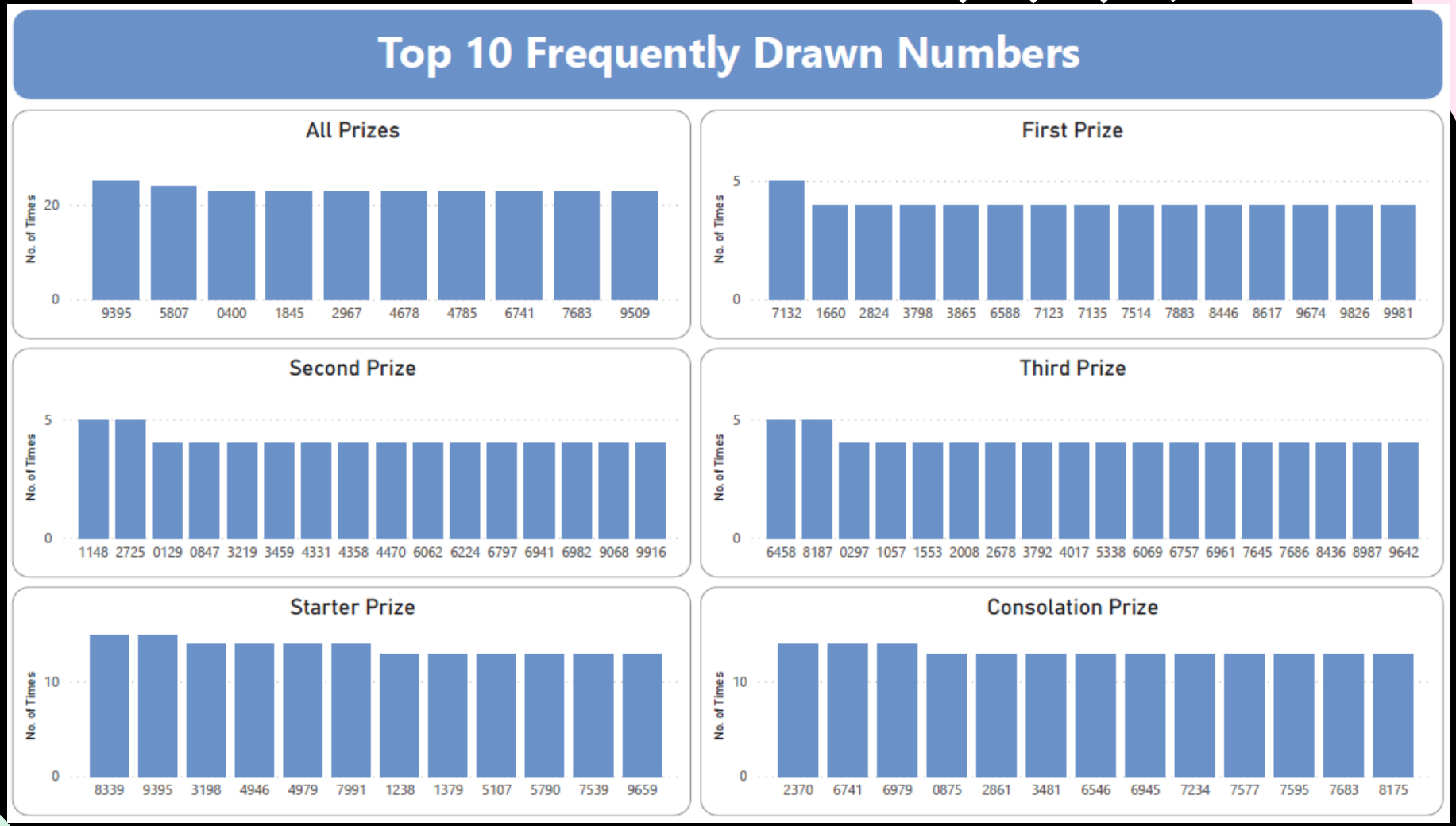
If you spent $4,798 on $1 Small for every draw in the past 35 years,

▲ a number in the Top 10 First Prize would have won you a total Prize Amount of at least $12,000

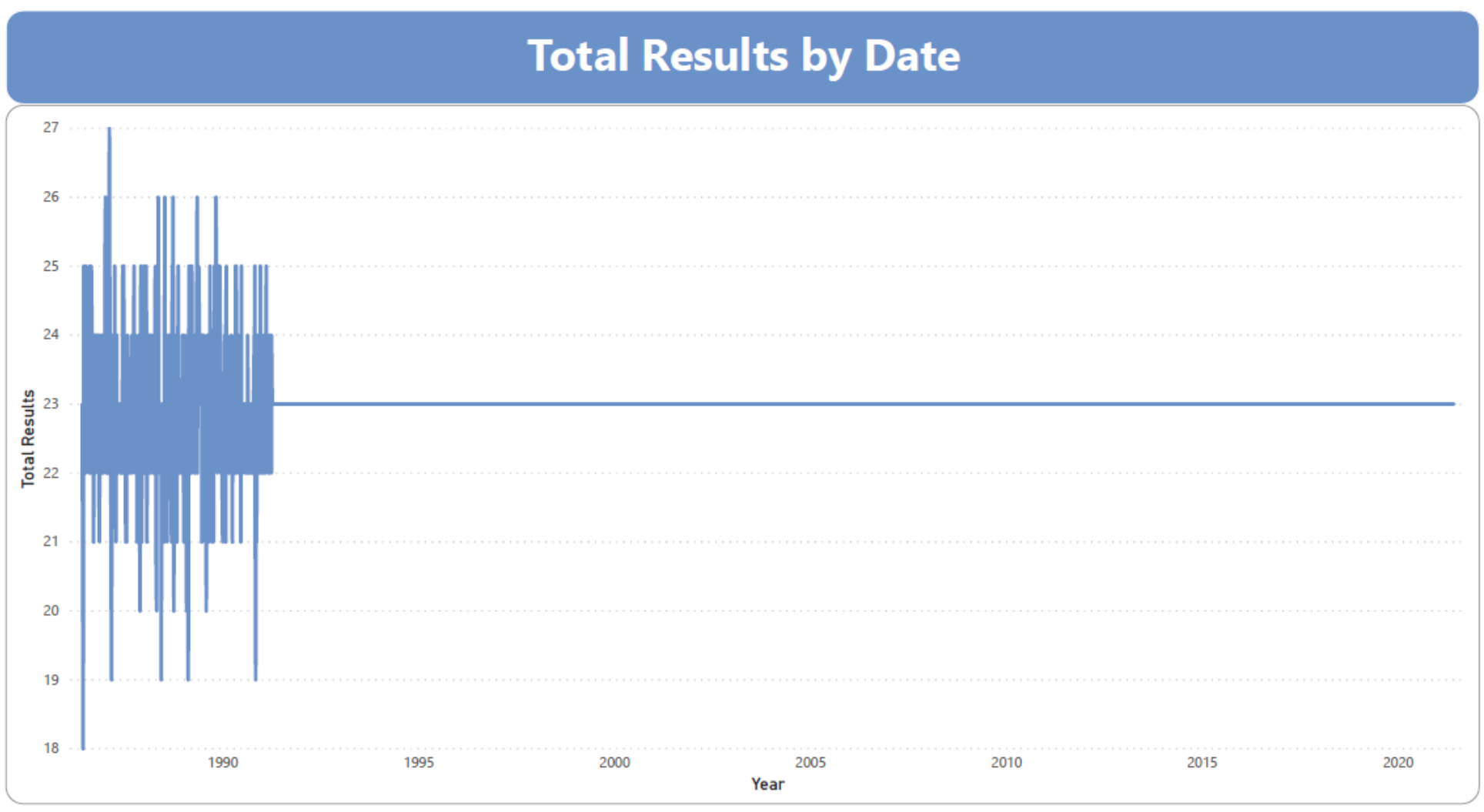
△ a number in the Top 10 Second Prize would have won you a total Prize Amount of at least $8,000

▽ a number in the Top 10 Third Prize would have won you a total Prize Amount of up to $4,000

Dashboard in Power BI (2)



Dashboard in Power BI (3)



Prior to April 1991, the number of Prizes per draw range from 18 to 27

Predictive Analytics (1)

1. **Predict all four digits ✗**

   Ordinary Entry: 0000 to 9999 (10,000 Classes)

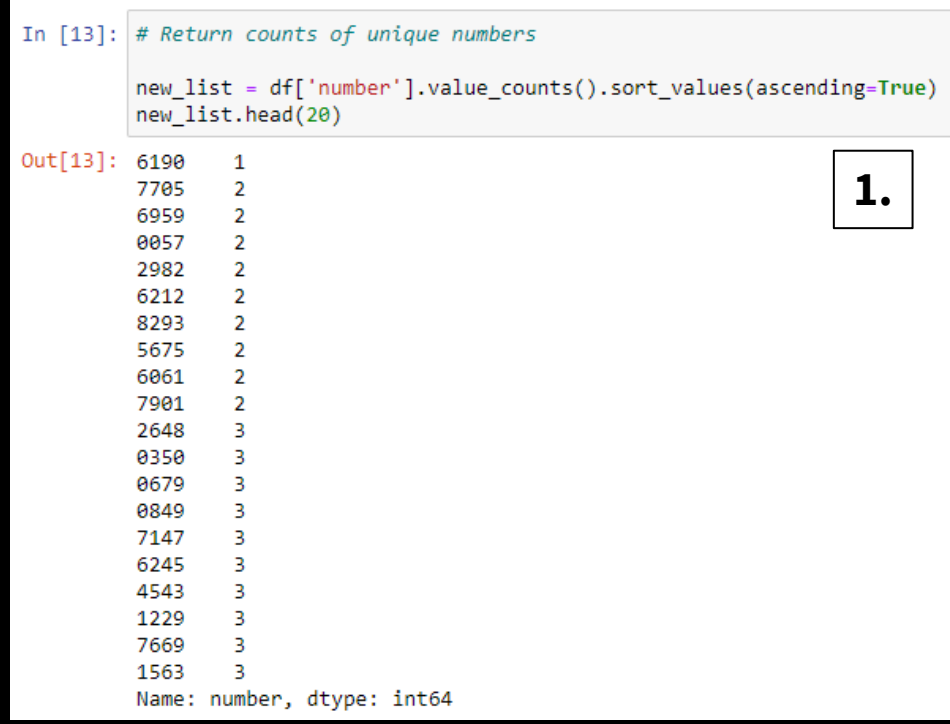
   Multiple classes have members less than cv=5 in the cross-validation splitting strategy

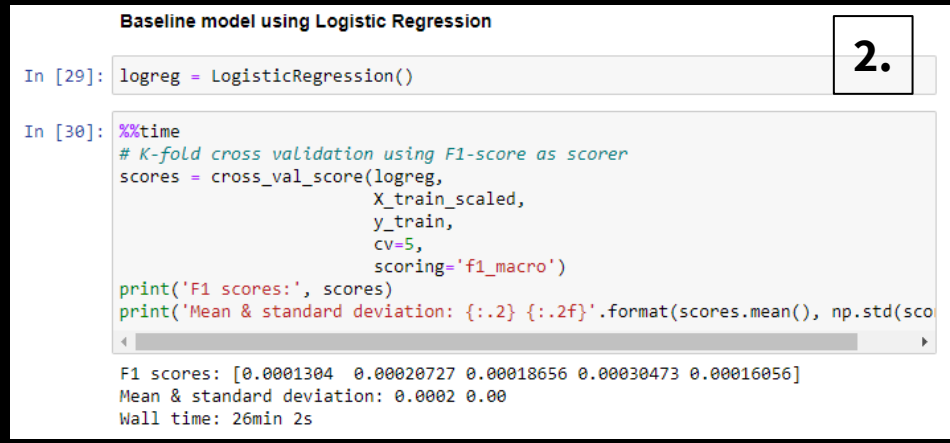
2. **Predict last three digits ✗**

   4D Roll: R000 to R999 (1,000 Classes)

   Low F1 scores (i.e. lower than 1/1,000) and takes very long to train the models

3. **Split the numbers into four and predict the first digit (10 Classes) ✓**





Predictive Analytics (2)

Models: (1) Logistic Regression
(2) K-Nearest Neighbors
(3) Multilayer Perceptron

Metrics: Balanced dataset hence accuracy can be used,
F1 and ROC-AUC scores were also derived

Tool: Python

Process Workflow

SOURCE: SINGAPORE POOLS

Data Pre-processing (1)

```
In [3]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110339 entries, 0 to 110338
Data columns (total 3 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   day     110339 non-null  object
 1   number  110339 non-null  int64
 2   prize   110339 non-null  object
dtypes: int64(1), object(2)
memory usage: 2.5+ MB
```

- Convert the 'number' column from numeric to fixed length string type (i.e. 4 digits)
- Split the 'day' column into 'day of week' and 'date' columns
- Convert the 'date' column to datetime
- Filter the 'date' column to remove data on 2021-6-6 due to partial scraping on that day
- Replace 'Thu' with 'Wed’ (only 1 draw on Thu, 25 Jan 2001, which replaced the draw on Wed, 24 Jan 2001)

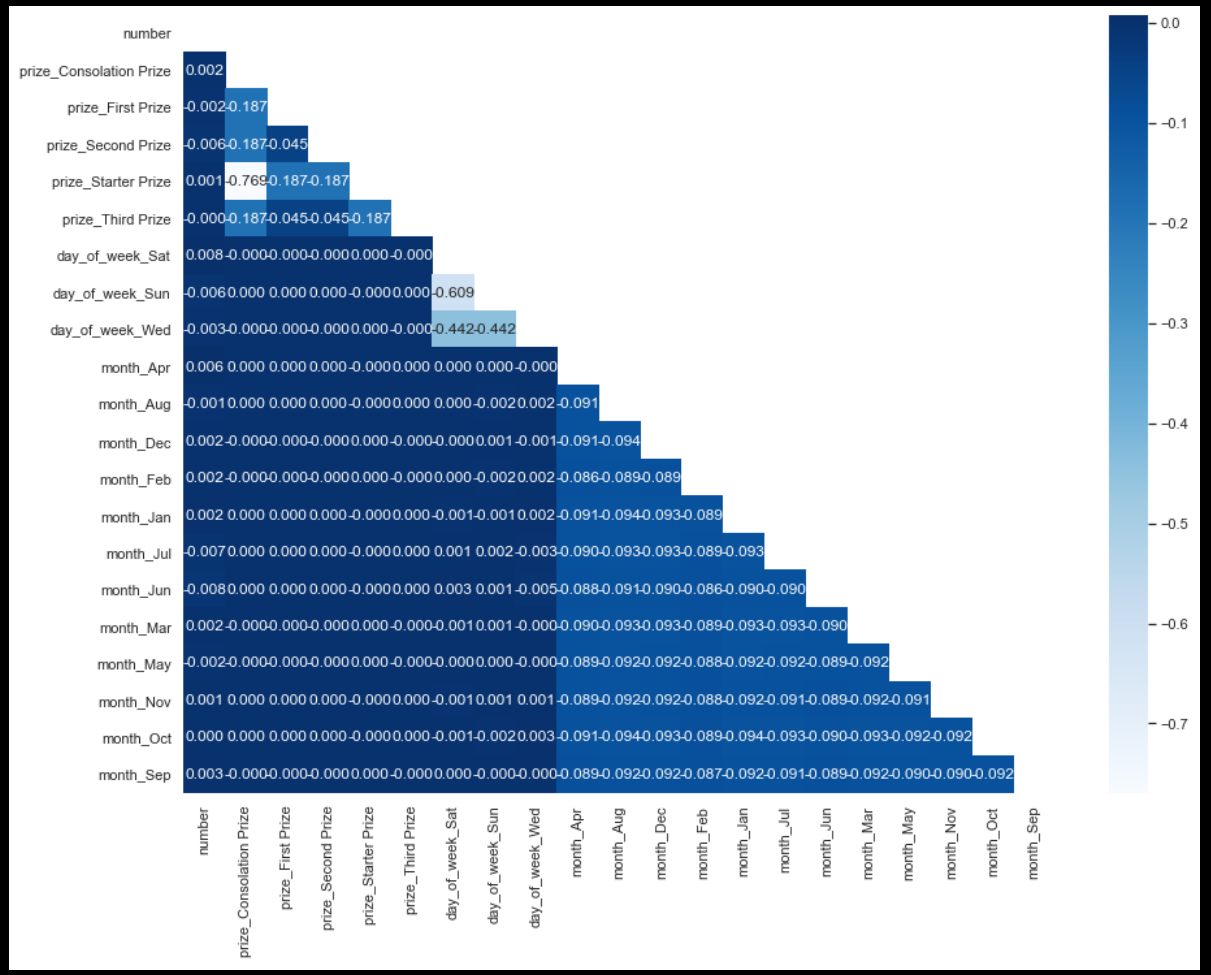
Data Pre-processing (2)

- Convert the 'date' column to the month abbreviated name format under 'month' column
- Split the ‘number’ column into 4 digits
- Use first digit as Class (from 10,000 classes reduced to 10 classes)
- Drop the redundant columns
- Encode variables with more than 2 Classes using one-hot encoding
- Convert the ‘num1' column to numeric

```
In [26]: df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 110322 entries, 0 to 110338
Data columns (total 45 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   prize_First Prize    110322 non-null  uint8
 1   prize_Second Prize   110322 non-null  uint8
 2   prize_Starter Prize  110322 non-null  uint8
 3   prize_Third Prize    110322 non-null  uint8
 4   day_of_week_Sun      110322 non-null  uint8
 5   day_of_week_Wed      110322 non-null  uint8
 6   month_Aug            110322 non-null  uint8
 7   month_Dec            110322 non-null  uint8
 8   month_Feb            110322 non-null  uint8
 9   month_Jan            110322 non-null  uint8
 10  month_Jul            110322 non-null  uint8
 11  month_Jun            110322 non-null  uint8
 12  month_Mar            110322 non-null  uint8
 13  month_May            110322 non-null  uint8
 14  month_Nov            110322 non-null  uint8
 15  month_Oct            110322 non-null  uint8
 16  month_Sep            110322 non-null  uint8
 17  num1                 110322 non-null  int64
 18  num2_1               110322 non-null  uint8
 19  num2_2               110322 non-null  uint8
 20  num2_3               110322 non-null  uint8
 21  num2_4               110322 non-null  uint8
 22  num2_5               110322 non-null  uint8
 23  num2_6               110322 non-null  uint8
 24  num2_7               110322 non-null  uint8
 25  num2_8               110322 non-null  uint8
 26  num2_9               110322 non-null  uint8
 27  num3_1               110322 non-null  uint8
 28  num3_2               110322 non-null  uint8
 29  num3_3               110322 non-null  uint8
 30  num3_4               110322 non-null  uint8
 31  num3_5               110322 non-null  uint8
 32  num3_6               110322 non-null  uint8
 33  num3_7               110322 non-null  uint8
 34  num3_8               110322 non-null  uint8
 35  num3_9               110322 non-null  uint8
 36  num4_1               110322 non-null  uint8
 37  num4_2               110322 non-null  uint8
 38  num4_3               110322 non-null  uint8
 39  num4_4               110322 non-null  uint8
 40  num4_5               110322 non-null  uint8
 41  num4_6               110322 non-null  uint8
 42  num4_7               110322 non-null  uint8
 43  num4_8               110322 non-null  uint8
 44  num4_9               110322 non-null  uint8
dtypes: int64(1), uint8(44)
memory usage: 6.3 MB
```

Exploratory Data Analysis (Correlation Matrix)



No correlation!

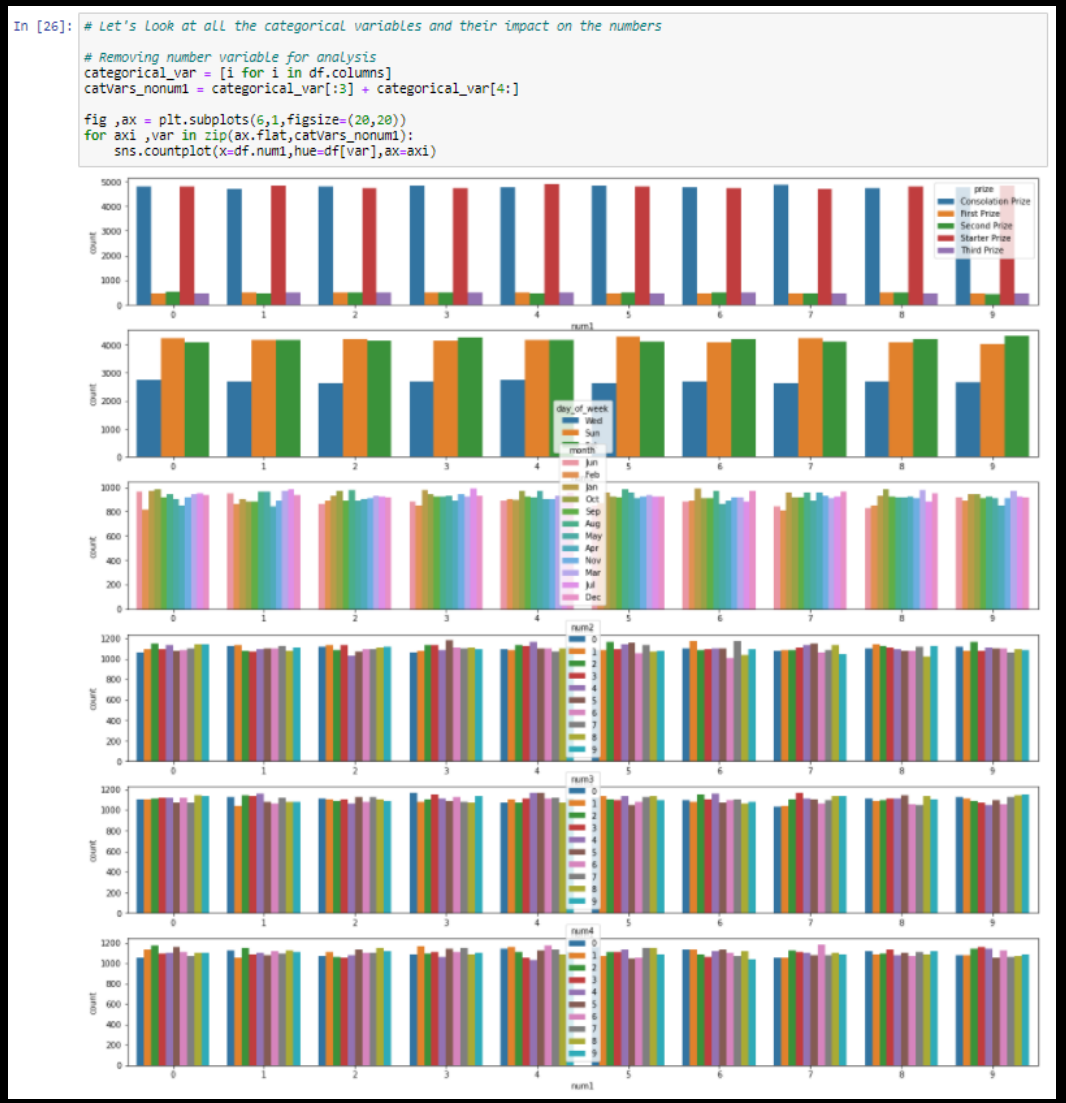
Exploratory Data Analysis (Count Plot)

Balanced dataset

In [26]:
```
# Let's look at all the categorical variables and their impact on the numbers

# Removing number variable for analysis
categorical_var = [i for i in df.columns]
catVars_nonum1 = categorical_var[:3] + categorical_var[4:]

fig ,ax = plt.subplots(6,1,figsize=(20,20))
for axi ,var in zip(ax.flat,catVars_nonum1):
    sns.countplot(x=df.num1,hue=df[var],ax=axi)
```



Machine Learning Baseline Model

Baseline model using Logistic Regression

In [29]:
```
logreg = LogisticRegression()
```

In [30]:
```
%%time
# K-fold cross validation using F1-score as scorer
scores = cross_val_score(logreg,
                         X_train_scaled,
                         y_train,
                         cv=5,
                         scoring='f1_macro')
print('F1 scores:', scores)
print('Mean & standard deviation: {:.2} {:.2f}'.format(scores.mean(), np.std(sco
```

```
F1 scores: [0.0997612  0.09558531 0.10202148 0.09915637 0.09973817]
Mean & standard deviation: 0.099 0.00
Wall time: 6.56 s
```

Hyperparameter Tuning (Logistic Regression)

In [32]:
```
# Logistic Regression
logreg = LogisticRegression(n_jobs=-1)
```

In [33]:
```
%%time

# Hyperparameter tuning using K-fold cross validation
# ... via Grid Search method
param_grid = {'penalty': ['l1', 'l2', 'elasticnet', 'none'],
              'C': np.logspace(-2, 2, 5)}

gs_logreg = GridSearchCV(logreg,
                         param_grid,
                         cv=5,
                         scoring='f1_macro',
                         n_jobs=-1)
gs_logreg.fit(X_train_scaled, y_train)
```

```
Wall time: 41.3 s
```

Out[33]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(n_jobs=-1), n_jobs=-1,
             param_grid={'C': array([1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02]),
                         'penalty': ['l1', 'l2', 'elasticnet', 'none']},
             scoring='f1_macro')
```

In [34]:
```
# Best model hyperparameters and score
print(gs_logreg.best_estimator_)
print(gs_logreg.best_params_)
print(gs_logreg.best_score_)
```

```
LogisticRegression(C=0.1, n_jobs=-1)
{'C': 0.1, 'penalty': 'l2'}
0.09930785805459272
```

Hyperparameter Tuning (K-Nearest Neighbors)

In [39]:
```
# K-Nearest Neighbors
classifier = KNeighborsClassifier()
```

In [40]:
```
%%time
# Hyperparameter tuning using K-fold cross validation
# ... via Grid Search method
parameters = {'n_neighbors': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]}

gs_clf = GridSearchCV(classifier,
                      parameters,
                      cv=5,
                      scoring='f1_macro',
                      n_jobs=-1)
gs_clf.fit(X_train, y_train)
```

```
Wall time: 1h 25min 9s
```

Out[40]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(), n_jobs=-1,
             param_grid={'n_neighbors': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]},
             scoring='f1_macro')
```

In [41]:
```
# Best model hyperparameters and score
print(gs_clf.best_estimator_)
print(gs_clf.best_params_)
print(gs_clf.best_score_)
```

```
KNeighborsClassifier(n_neighbors=8)
{'n_neighbors': 8}
0.09812296990860483
```

Hyperparameter Tuning (Multilayer Perceptron)

In [46]:
```
# Multilayer Perceptron (stochastic iterative)
mlp = MLPClassifier(solver='sgd')
```

In [47]:
```
%%time
# Hyperparameter tuning using K-fold cross validation
# ... via Grid Search method
param_grid = {'hidden_layer_sizes': [(3,3),
                                     (2),
                                     (3)],
              'alpha': np.logspace(-4, -1, 4),
              'max_iter': [400, 450, 500, 550]}

gs_mlp = GridSearchCV(mlp,
                      param_grid,
                      cv=5,
                      scoring='f1_macro',
                      n_jobs=-1)
gs_mlp.fit(X_train_scaled, y_train)
```

```
Wall time: 11min 58s
```

Out[47]:
```
GridSearchCV(cv=5, estimator=MLPClassifier(solver='sgd'), n_jobs=-1,
             param_grid={'alpha': array([0.0001, 0.001 , 0.01  , 0.1   ]),
                         'hidden_layer_sizes': [(3, 3), 2, 3],
                         'max_iter': [400, 450, 500, 550]},
             scoring='f1_macro')
```

In [48]:
```
# Best model hyperparameters and score
print(gs_mlp.best_estimator_)
print(gs_mlp.best_params_)
print(gs_mlp.best_score_)
```

```
MLPClassifier(alpha=0.1, hidden_layer_sizes=3, max_iter=400, solver='sgd')
{'alpha': 0.1, 'hidden_layer_sizes': 3, 'max_iter': 400}
0.07089168239629182
```

Machine Learning Model Selection

	Model	Training Acc	Testing Acc	Precision	Recall	F1 Score
0	Logistic Regression	0.111505	0.101700	0.101467	0.101666	0.101023
1	K-Nearest Neighbors	0.302269	0.098002	0.096530	0.098041	0.093998
2	Multilayer Perceptron	0.104942	0.098184	0.081481	0.098329	0.059731

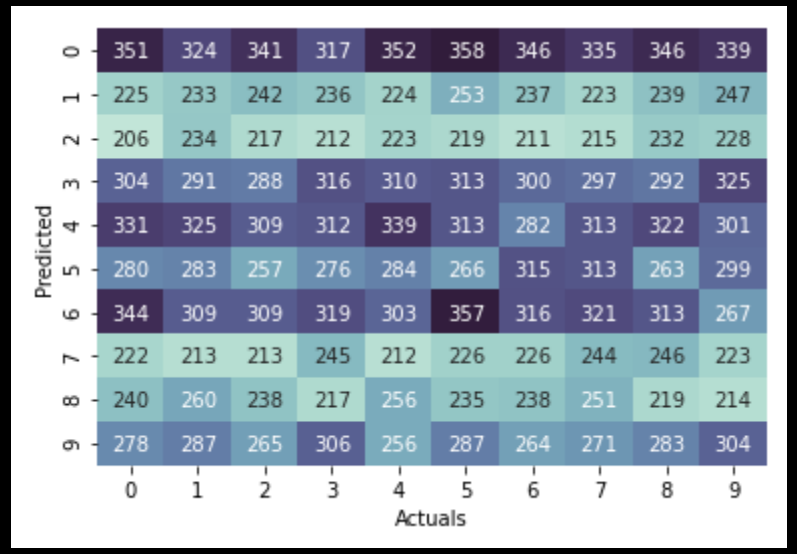
Select Logistic Regression...

Results

Classification Report

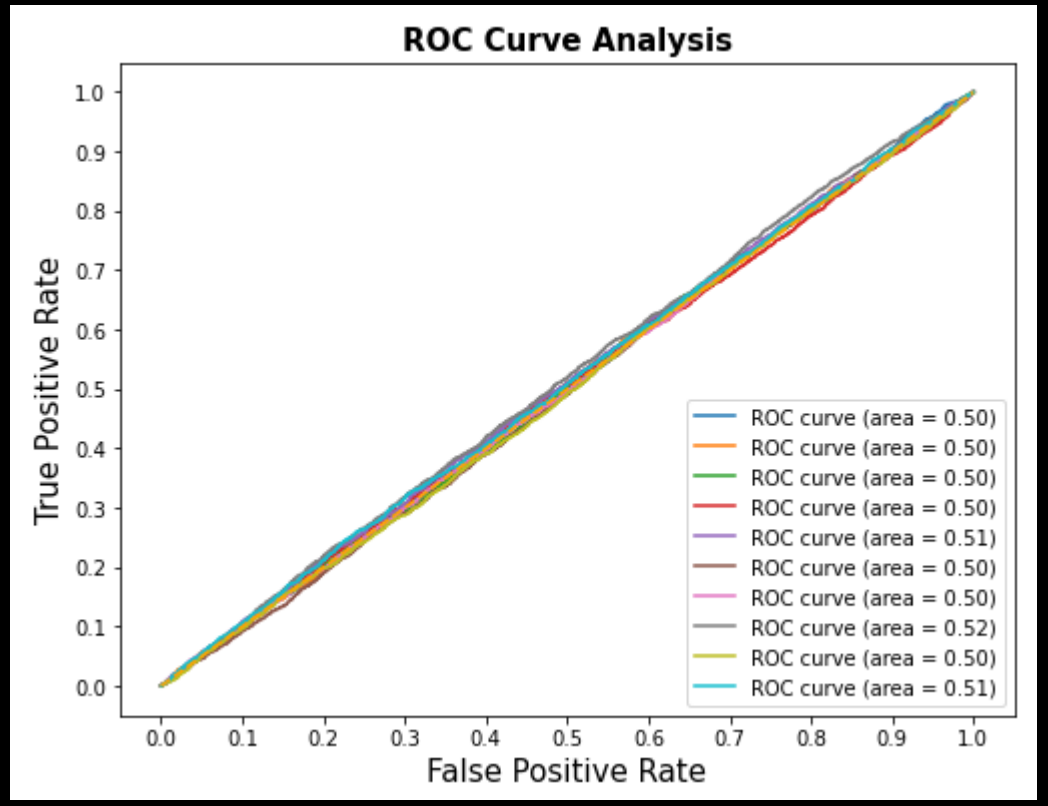
	precision	recall	f1-score	support
0	0.10	0.13	0.11	2781
1	0.10	0.08	0.09	2759
2	0.10	0.08	0.09	2679
3	0.10	0.11	0.11	2756
4	0.11	0.12	0.11	2759
5	0.09	0.09	0.09	2827
6	0.10	0.12	0.11	2735
7	0.11	0.09	0.10	2783
8	0.09	0.08	0.09	2755
9	0.11	0.11	0.11	2747
accuracy			0.10	27581
macro avg	0.10	0.10	0.10	27581
weighted avg	0.10	0.10	0.10	27581

Confusion Matrix



- **Low F1 score of 0.10**
- **Low accuracy of 0.10**
- **1/10 chance of predicting the number correctly (i.e. 0 to 9)**

Receiver Operating Characteristic Curve



Not healthy

Conclusions

Conclusions

- **Can we predict 4D results?**
- **Can we predict the number on a given day?**
- **Can we predict which prize category the number will be?**

The answers are no, no and no.

Thirty-five years worth of data have shown that some numbers are drawn more frequently than others.

However, the fundamental truth is that the theoretical probability of every number is the same (i.e. 1/10,000).

We are looking for patterns where non exists.

Q&A