03_Python_기초정리.ipynb

to_csv 에러 없애는 방법 : errors

In [3]:
```
df.to_csv('test.csv', encoding='cp949', errors='replace')   # 'replace', 'ignore'
```

DataFrame 행 추가

In [ ]:
```
new_data = {'성별코드': '3', '연령대코드(5세단위)' : 10, '허리둘레':100, '흡연상태': 1}
df.append(new_data, ignore_index=True)
```

statsmodels 패키지 사용법

Dummy화 쉽게 하는 법

In [44]:
```
import statsmodels.api as sm
import pandas
from patsy import dmatrices

df = sm.datasets.get_rdataset("Guerry", "HistData").data
vars = ['Department', 'Lottery', 'Literacy', 'Wealth', 'Region']
df = df[vars]
df = df.dropna()
df.head()
```

Out[44]:

	Department	Lottery	Literacy	Wealth	Region
0	Ain	41	37	73	E
1	Aisne	38	51	22	N
2	Allier	66	13	61	C
3	Basses-Alpes	80	46	76	E
4	Hautes-Alpes	79	69	83	E

In [45]:
```
y, X = dmatrices('Lottery ~ Literacy + Wealth + C(Region) + Literacy:Wealth', data=df, return_type='dataframe')
```

In [48]:
```
X.head()   ## 범주형 변수(Region)에서 NaN 값은 drop해주고, class 중 하나(여기서는 C)를 drop하고 나머지는 더미화
```

Out[48]:

	Intercept	Region[T.E]	Region[T.N]	Region[T.S]	Region[T.W]	Literacy	Wealth
0	1.0	1.0	0.0	0.0	0.0	37.0	73.0
1	1.0	0.0	1.0	0.0	0.0	51.0	22.0
2	1.0	0.0	0.0	0.0	0.0	13.0	61.0
3	1.0	1.0	0.0	0.0	0.0	46.0	76.0
4	1.0	1.0	0.0	0.0	0.0	69.0	83.0

Fit & Summary

- 1단계: 알고리즘 선택
- 2단계: Fit
- 3단계: Summarize

In [49]:
```
# OLS(최소자승법) 회귀분석을 예시로 함

mod = sm.OLS(y, X) # 알고리즘 선택
res = mod.fit() # Fit
print(res.summary()) # Summarize model
```

OLS Regression Results

Dep. Variable:	Lottery	R-squared:	0.338
Model:	OLS	Adj. R-squared:	0.287
Method:	Least Squares	F-statistic:	6.636
Date:	Fri, 04 Dec 2020	Prob (F-statistic):	1.07e-05
Time:	22:00:34	Log-Likelihood:	-375.30
No. Observations:	85	AIC:	764.6
Df Residuals:	78	BIC:	781.7
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	38.6517	9.456	4.087	0.000	19.826	57.478
Region[T.E]	-15.4278	9.727	-1.586	0.117	-34.793	3.938
Region[T.N]	-10.0170	9.260	-1.082	0.283	-28.453	8.419
Region[T.S]	-4.5483	7.279	-0.625	0.534	-19.039	9.943
Region[T.W]	-10.0913	7.196	-1.402	0.165	-24.418	4.235
Literacy	-0.1858	0.210	-0.886	0.378	-0.603	0.232
Wealth	0.4515	0.103	4.390	0.000	0.247	0.656

Omnibus:	3.049	Durbin-Watson:	1.785
Prob(Omnibus):	0.218	Jarque-Bera (JB):	2.694
Skew:	-0.340	Prob(JB):	0.260
Kurtosis:	2.454	Cond. No.	371.

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [50]:
```
# 변수별 계수 추출
res.params
```

Out[50]:
```
Intercept      38.651655
Region[T.E]   -15.427785
Region[T.N]   -10.016961
Region[T.S]    -4.548257
Region[T.W]   -10.091276
Literacy       -0.185819
Wealth          0.451475
dtype: float64
```

In [51]:
```
# r-squared
res.rsquared
```

Out[51]:
```
0.3379508691928822
```

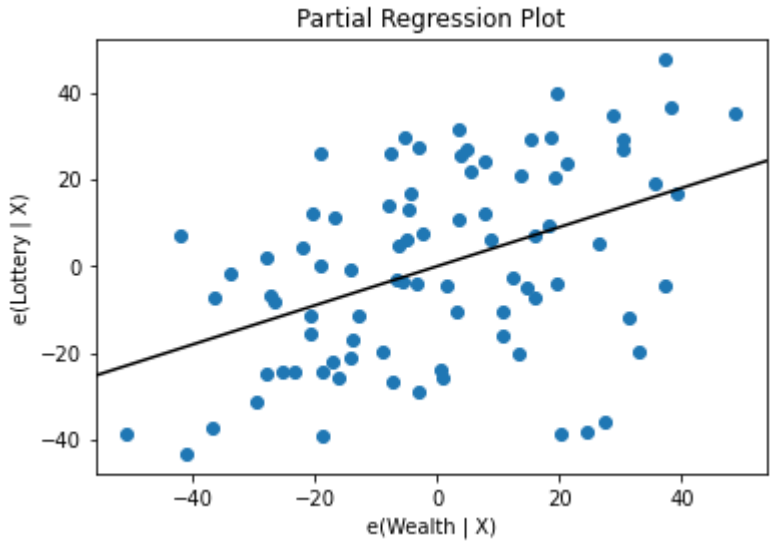
In [ ]:
```
# 사용 가능한 attriibutes 확인
dir(res)
```

Diagnostics and specification tests

In [57]:
```
sm.stats.linear_rainbow(res)
```

Out[57]:
```
(0.847233997615691, 0.6997965543621644)
```

In [56]:
```
%matplotlib inline
sm.graphics.plot_partregress('Lottery', 'Wealth', ['Region', 'Literacy'], data
=df.dropna(), obs_labels=False)
plt.show()
```



In [ ]:
```
```

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

data = sm.datasets.scotland.load(as_pandas=False)
data.exog = sm.add_constant(data.exog)
```

In [63]:
```
# y 값 형태
data.endog
```

Out[63]:
```
array([60.3, 52.3, 53.4, 57. , 68.7, 48.8, 65.5, 70.5, 59.1, 62.7, 51.6,
       62. , 68.4, 69.2, 64.7, 75. , 62.1, 67.2, 67.7, 52.7, 65.7, 72.2,
       47.4, 51.3, 63.6, 50.7, 51.6, 56.2, 67.6, 58.9, 74.7, 67.3])
```

In [ ]:
```
# X 값 형태
data.exog
```

In [67]:
```
# Linear Regression
OLS(endog[, exog, missing, hasconst]) # Ordinary Least Squares
GLS(endog, exog[, sigma, missing, hasconst]) # Generalized Least Squares

# Logistic Regression
gamma_model = sm.GLM(data.endog, data.exog, family=sm.families.Gamma())
gamma_results = gamma_model.fit()
print(gamma_results.summary())
```

OLS Regression Results

Dep. Variable:	y	R-squared:	0.842
Model:	OLS	Adj. R-squared:	0.795
Method:	Least Squares	F-statistic:	18.21
Date:	Fri, 04 Dec 2020	Prob (F-statistic):	3.54e-08
Time:	22:13:57	Log-Likelihood:	-81.844
No. Observations:	32	AIC:	179.7
Df Residuals:	24	BIC:	191.4
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	137.4141	40.922	3.358	0.003	52.956	221.872
x1	-0.1165	0.058	-2.009	0.056	-0.236	0.003
x2	-5.1860	1.849	-2.805	0.010	-9.002	-1.370
x3	0.2846	0.098	2.896	0.008	0.082	0.487
x4	-0.4204	0.158	-2.659	0.014	-0.747	-0.094
x5	0.0005	0.000	1.041	0.308	-0.000	0.001
x6	1.8404	0.891	2.065	0.050	0.001	3.680
x7	0.0059	0.003	2.259	0.033	0.001	0.011

Omnibus:	2.108	Durbin-Watson:	1.654
Prob(Omnibus):	0.348	Jarque-Bera (JB):	1.281
Skew:	-0.483	Prob(JB):	0.527
Kurtosis:	3.170	Cond. No.	1.26e+06

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
[2] The condition number is large, 1.26e+06. This might indicate that there a
re
strong multicollinearity or other numerical problems.
```

In [ ]:

In [ ]:

06_통계분석_1

범주형 변수 빈도분석

In [ ]:
```
# 범주형 변수 빈도분석

da_cat = pd.DataFrame()
for i in  col_cat:
    a = df[i].value_counts(dropna =False).to_frame().sort_index().rename(colum
ns={i:'count'}).reset_index()
    a['col_nm'] = i
    a = a.rename(columns = {'index':'class'})
    a = a[['col_nm','class','count']]
    b = df[i].value_counts(dropna =False, normalize = True).to_frame().sort_in
dex().rename(columns={i:'ratio'}).reset_index()
    b = b['ratio'].to_frame()
    a = pd.concat([a,b],axis=1)
    da_cat = pd.concat([da_cat,a], axis = 0)
da_cat = da_cat.reset_index(drop=True)
da_cat
```

교호작용 확인

- 한 변수에 의해 다른 변수의 효과가 변하는 것을 말합니다.

다중공선성 제거

- 다중공선성은 예측변수 사이에 높은 상관관계가 있을 때 발생하며, 회귀계수 추정치의 신뢰성과 안정성에 문제를 발생시킨다.
- 다중공선성을 진단하는데 널리 사용되는 분산팽창계수(variance inflation factor, VIF)
- VIF가 1.8이라는 것은 특정 회귀계수의 분산(표준오차의 제곱)이 만약 해당 예측변수가 나머지 예측변수와 완전히 상관관계가 없다면 가졌을 분산보다 80% 크다는 의미이다. VIF는 하한선이 1이지만 상한선이 없다.
- VIF가 10이 넘어가면 다중공선성이 있다고 판단하여 제거한다(변수를 하나씩 선택 제거해가며 VIF 계속 확인)
- 분석상 논리적으로 필요하다고 판단되면 유지해본다

In [ ]:
```
import statsmodels.api as sm
import pandas
from patsy import dmatrices

y, X = dmatrices('y2 ~ season + holiday', data=df, return_type='dataframe')

from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(X.values, i) for i in range(X.s
hape[1])]
vif["features"] = X.columns
vif
```

회귀분석 - SVM

- LinearSVR(C=1.0, dual=True, epsilon=1.5, fit_intercept=True, intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=1000, random_state=42, tol=0.0001, verbose=0)

In [12]:
```
np.random.seed(42)
m = 50
X = 2 * np.random.rand(m,1)
y = (4+3*X+np.random.randn(m,1)).ravel()
```

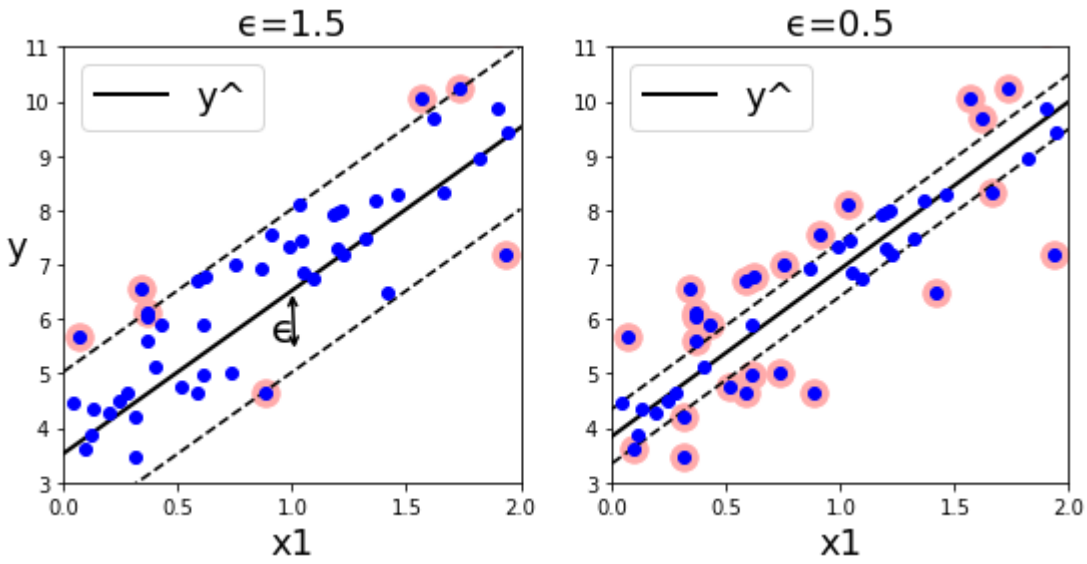
In [13]:
```
from sklearn.svm import LinearSVR

svm_reg =LinearSVR(epsilon=1.5, random_state=42) # svm_reg = 기본 모형
svm_reg1 = LinearSVR(epsilon=1.5, random_state=42) # svm_reg1 = 마진이 큰 모형(e
psilon=1.5)
svm_reg2 = LinearSVR(epsilon=0.5, random_state=42) # svm_reg2 = 마진이 작은 모형
(epsilon=0.5)

svm_reg.fit(X,y)
svm_reg1.fit(X,y)
svm_reg2.fit(X,y)
```

Out[13]:
```
LinearSVR(epsilon=0.5, random_state=42)
```

In [14]:
```
# 서포트 벡터 정하기
#svm_reg 모델을 받고, X로 예측한 값 = y_pred
#off_margin = 실제 y값과 예측값 사이의 오차를 절대값으로 표현하되, 해당 모형의 epsilon보다 크거나 같은 값
#np.argwhere는 행렬에서 True에 해당하는 값 위치를 반환합니다.
#따라서, 오차가 epsilon보다 큰 값들의 위치 반환하며 이 것들이 곧 서포트 벡터로 활용
def find_support_vectors(svm_reg, X, y):
    y_pred = svm_reg.predict(X)
    off_margin = (np.abs(y - y_pred) >= svm_reg.epsilon)
    return np.argwhere(off_margin)

svm_reg1.support_ = find_support_vectors(svm_reg1, X, y)
svm_reg2.support_ = find_support_vectors(svm_reg2, X, y)

eps_x1 = 1
eps_y_pred = svm_reg1.predict([[eps_x1]])
```

In [15]:
```
#plot찍기
#np.linspace로 axes의 첫 번째 값과 두 번째 값 사이를 100개로 쪼갠 일정한 값 생성 후 100행 1열로 reshape
#위 값(x1s)으로 예측한 y = y_pred
#x, y를 plot하고, y_pred에서 epsilon을 빼고 더한 값도 plot
#아까 구했던 서포트 벡터(지지도 벡터)를 scatter 찍기
def plot_svm_regression(svm_reg, X, y, axes) :
    x1s = np.linspace(axes[0], axes[1], 100).reshape(100,1)
    y_pred = svm_reg.predict(x1s)
    plt.plot(x1s, y_pred, 'k-', linewidth=2, label = r'y^')
    plt.plot(x1s, y_pred + svm_reg.epsilon, 'k--')
    plt.plot(x1s, y_pred - svm_reg.epsilon, 'k--')
    plt.scatter(X[svm_reg.support_], y[svm_reg.support_], s=180, facecolors='#FFAAAA')
    plt.plot(X, y, 'bo')
    plt.xlabel(r'x1', fontsize=18)
    plt.legend(loc='upper left', fontsize=18)
    plt.axis(axes)
```

In [16]:
```
plt.figure(figsize=(9, 4))
plt.subplot(121)
plot_svm_regression(svm_reg1, X, y, [0,2,3,11])
plt.title(r'ϵ={}'.format(svm_reg1.epsilon), fontsize=18)
plt.ylabel(r'y', fontsize=18, rotation=0)
#plt.plot([eps_x1, eps_x1], [eps_y_pred ,eps_y_pred - svm_reg1.epsilon], 'k-',
linewidth=2)

#위에서 eps_x1 = 1 로 두고, 예측한 값 = eps_y_pred
#화살표와 함께 text를 넘기는 annotate
#따라서 도로의 폭을 나타냅니다. (epsilon)
#xy 는 주석을 달 위치입니다. ( eps_x1, eps_y_pred )
#xytext 는 xy위치에 넣을 text
#textcoords 는 지금 넣은 'data'가 default값이며, 주석을 달 객체의 좌표값 사용을 뜻합니
다.
#arrowprops는 화살표설정

plt.annotate(
    ' ', xy = (eps_x1, eps_y_pred), xycoords='data',
    xytext = (eps_x1, eps_y_pred-svm_reg1.epsilon),
    textcoords='data', arrowprops={'arrowstyle': '<->', 'linewidth' : 1.5}
    )

plt.text(0.91, 5.6, r'ϵ', fontsize=20)
plt.subplot(122)
plot_svm_regression(svm_reg2, X, y, [0,2,3,11])
plt.title(r'ϵ={}'.format(svm_reg2.epsilon), fontsize=18)

plt.show()
```



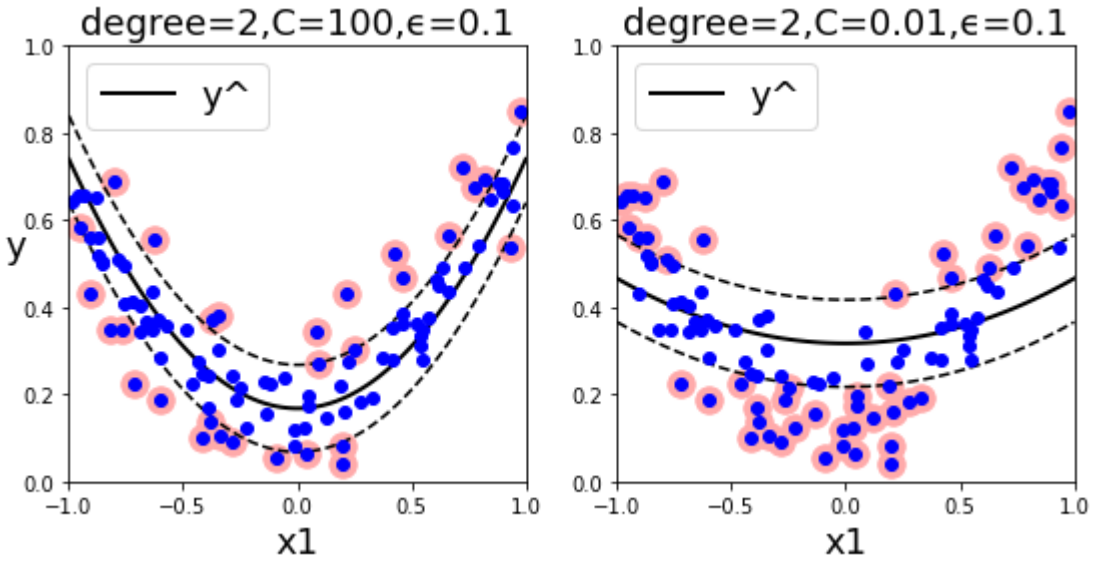
회귀분석 - SVM - 비선형

In [9]:
```
np.random.seed(42)
m = 100
X = 2* np.random.rand(m,1) -1
y = (0.2 + 0.1 * X + 0.5 * X**2 + np.random.randn(m,1) / 10).ravel()
```

In [10]:
```
from sklearn.svm import SVR

svm_poly_reg = SVR(kernel='poly', gamma='auto', degree=2, C=10, epsilon=0.1)
svm_poly_reg.fit(X,y)

svm_poly_reg1 = SVR(kernel='poly', gamma='auto', degree=2, C=100, epsilon=0.1)
svm_poly_reg2 = SVR(kernel='poly', gamma='auto', degree=2, C=0.01, epsilon=0.1
)
svm_poly_reg1.fit(X,y)
svm_poly_reg2.fit(X,y)
```

Out[10]:
```
SVR(C=0.01, degree=2, gamma='auto', kernel='poly')
```

In [11]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(9,4))

plt.subplot(121)
plot_svm_regression(svm_poly_reg1, X, y, [-1, 1, 0, 1])
plt.title(r'degree={},C={},ϵ={}'.format(svm_poly_reg1.degree, svm_poly_reg1.C,
svm_poly_reg1.epsilon), fontsize=18)

plt.ylabel(r'y', fontsize=18, rotation=0)

plt.subplot(122)
plot_svm_regression(svm_poly_reg2, X, y, [-1, 1, 0, 1])
plt.title(r'degree={},C={},ϵ={}'.format(svm_poly_reg2.degree, svm_poly_reg2.C,
svm_poly_reg2.epsilon), fontsize=18)

plt.show()
```



In [ ]:

06_통계분석_2

시계열 분석

- https://www.youtube.com/watch?v=rdR2fNDq6v0&ab_channel=%E2%80%8D%EA%B9%80%EC%84%B1%EB%B2%94%5B%EB%8B%A8%I (https://www.youtube.com/watch?v=rdR2fNDq6v0&ab_channel=%E2%80%8D%EA%B9%80%EC%84%B1%EB%B2%94%5B%EB%8B%A8%I

In [ ]:
```
# 정상성 확인 방법
# 그래프로 확인 방법(추세가 있거나, 계절성이 있거나, 분산이 변하면 안 됨)
# 평균과 분산이 시점에 관계없이 일정해야하고, 공분산도 시차에만 의존해야함
```

모듈 불러오기

In [2]:
```
import os

import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

import matplotlib.pyplot as plt
import matplotlib
plt.style.use('seaborn-whitegrid')

import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from pmdarima.arima import auto_arima  ## ADP 볼 때는 없을 패키지

import seaborn as sns

%matplotlib inline

import itertools # 내장 패키지
```

데이터 불러오기

- 활용 데이터 : 공공 데이터(AirPassengers.csv)
- 1949~1960년 매달 비행기 탑승객 수에 대한 시계열 데이터
- 데이터 구조 : [144 x 1]

In [3]:
```
data = pd.read_csv('./data/AirPassengers.csv')
data = data.rename(columns = {'Month':'month', '#Passengers':'passengers'})
data['month'] = pd.to_datetime(data['month'])
data = data.set_index('month')   ### 시간 컬럼을 인덱스로 만들어줌
data.head()
```

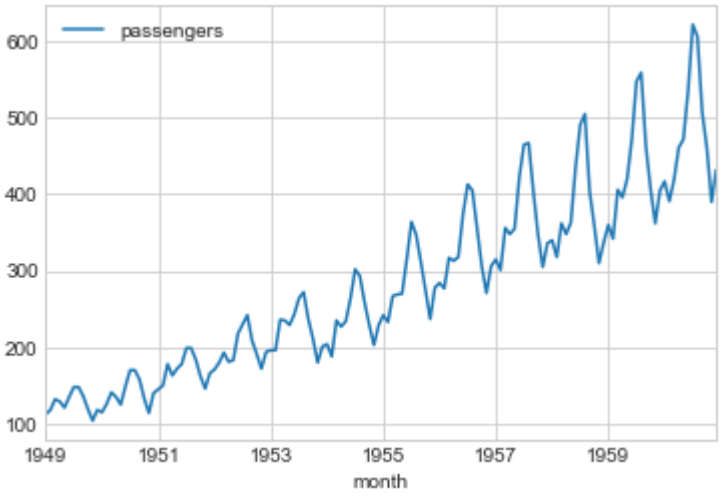
Out[3]:

month	passengers
1949-01-01	112
1949-02-01	118
1949-03-01	132
1949-04-01	129
1949-05-01	121

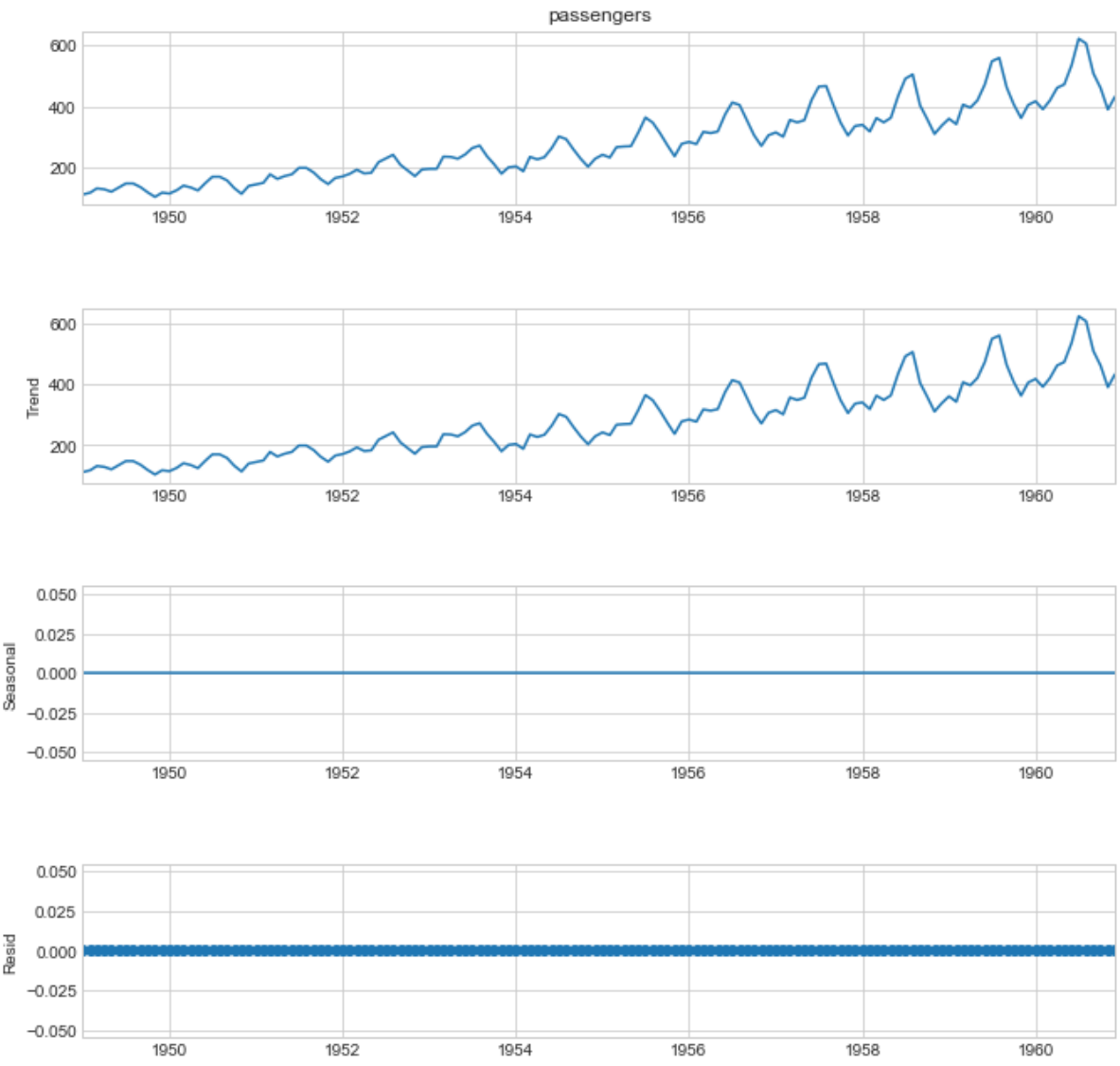
Box-Jenjins ARIMA Procedure

1) Data Preprocessing
2) Identify Model to be Tentatively Entertainted
3) Estimate Parameters
4) Diagnosis Check
5) Use Model to Forecast

In [3]:
```
fig = data.plot()
```



In [5]:
```
# Seasonal decomposition plot : Seasonal decomposition using moving averages.

# Observed : observed data
# Trend : The estimated trend component
# Seasonal : The estimated seasonal component
# resid : The estimated residuals
decomposition = sm.tsa.seasonal_decompose(data['passengers'], model = 'additiv
e', period=1)
fig = decomposition.plot()
fig.set_size_inches(10,10)
plt.show()
```



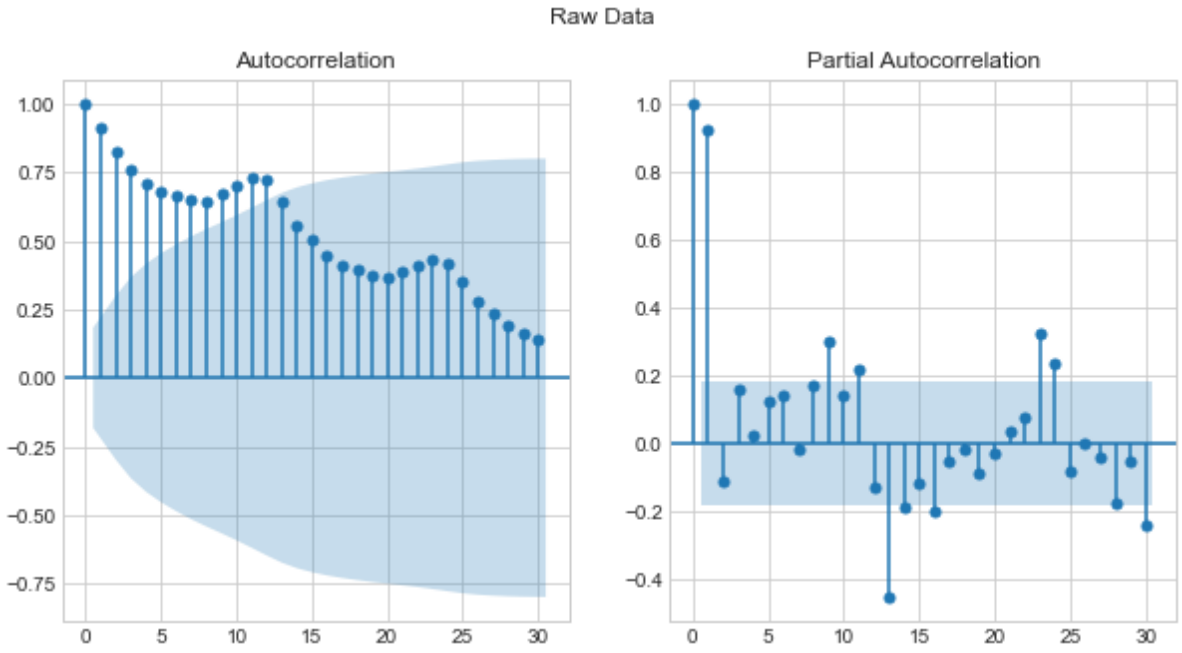
Identify Model to be Tentatively Entertainted

In [9]:
```
# Tr, Te = 8:2
train_data, test_data = train_test_split(data, test_size=0.2, shuffle=False)
```

In [10]:
```
# 참고 #
train_data.values.squeeze()
```

Out[10]:
```
array([112, 118, 132, 129, 121, 135, 148, 148, 136, 119, 104, 118, 115,
       126, 141, 135, 125, 149, 170, 170, 158, 133, 114, 140, 145, 150,
       178, 163, 172, 178, 199, 199, 184, 162, 146, 166, 171, 180, 193,
       181, 183, 218, 230, 242, 209, 191, 172, 194, 196, 196, 236, 235,
       229, 243, 264, 272, 237, 211, 180, 201, 204, 188, 235, 227, 234,
       264, 302, 293, 259, 229, 203, 229, 242, 233, 267, 269, 270, 315,
       364, 347, 312, 274, 237, 278, 284, 277, 317, 313, 318, 374, 413,
       405, 355, 306, 271, 306, 315, 301, 356, 348, 355, 422, 465, 467,
       404, 347, 305, 336, 340, 318, 362, 348, 363, 435, 491], dtype=int64)
```

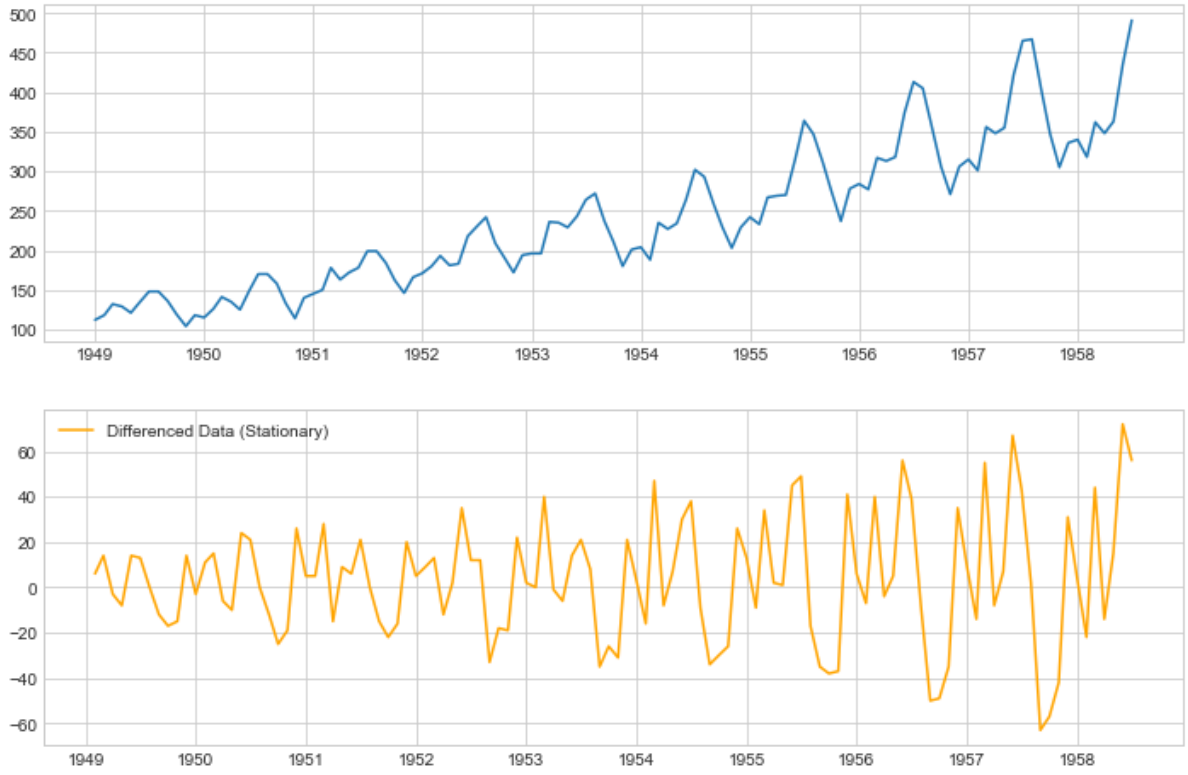
In [11]:
```
# ACF, PACF plot

fig, ax = plt.subplots(1,2, figsize = (10, 5))
fig.suptitle('Raw Data')
sm.graphics.tsa.plot_acf(train_data.values.squeeze(), lags = 30, ax = ax[0])
sm.graphics.tsa.plot_pacf(train_data.values.squeeze(), lags = 30, ax = ax[1])
plt.show()
### ACF 그래프가 점진적으로 감소하는 것은 전형적인 Non-stationary 데이터이다 = 정상정이
 없음
```



In [12]:
```
# Differencing

diff_train_data = train_data.copy()
diff_train_data = diff_train_data['passengers'].diff()
diff_train_data = diff_train_data.dropna()
print('##### Raw Data #####')
print(train_data)
print('### Differenced Data ###')
print(diff_train_data)
```

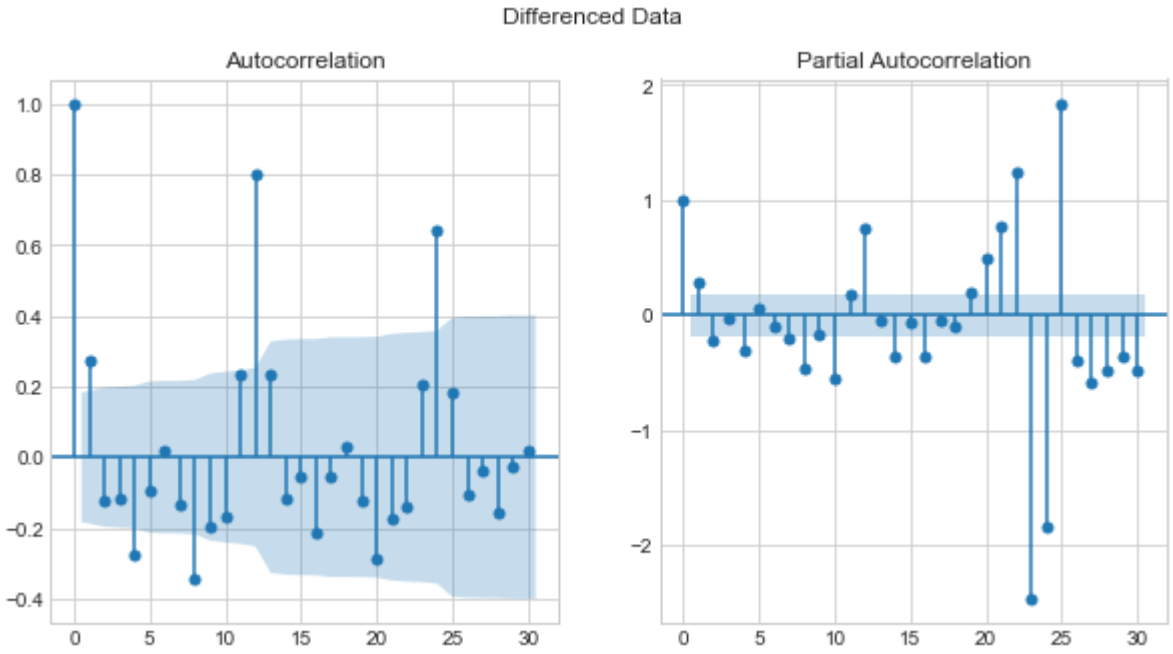
```
##### Raw Data #####
            passengers
month
1949-01-01         112
1949-02-01         118
1949-03-01         132
1949-04-01         129
1949-05-01         121
...                ...
1958-03-01         362
1958-04-01         348
1958-05-01         363
1958-06-01         435
1958-07-01         491

[115 rows x 1 columns]
### Differenced Data ###
month
1949-02-01     6.0
1949-03-01    14.0
1949-04-01    -3.0
1949-05-01    -8.0
1949-06-01    14.0
              ...
1958-03-01    44.0
1958-04-01   -14.0
1958-05-01    15.0
1958-06-01    72.0
1958-07-01    56.0
Name: passengers, Length: 114, dtype: float64
```

In [13]:
```
# differenced data plot

plt.figure(figsize = (12,8))
plt.subplot(211)
plt.plot(train_data['passengers'])
plt.subplot(212)
plt.plot(diff_train_data, 'orange') # first difference (t - (t-1))
plt.legend(['Differenced Data (Stationary)'])
plt.show()
```



In [14]:
```
# ACF, PACF plot

fig, ax = plt.subplots(1,2, figsize = (10, 5))
fig.suptitle('Differenced Data')
sm.graphics.tsa.plot_acf(diff_train_data.values.squeeze(), lags = 30, ax = ax[
0])
sm.graphics.tsa.plot_pacf(diff_train_data.values.squeeze(), lags = 30, ax = ax
[1])
plt.show()
```

```
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\statsmo
dels\regression\linear_model.py:1406: RuntimeWarning: invalid value encounter
ed in sqrt
  return rho, np.sqrt(sigmasq)
```



- ACF는 싸인 곡선의 형태로 점차 감소한다고 보이고, PACF는 애매하지만 Lag 1 이후 절단면을 가지는 것으로 판단하여 AR(1) 모델을 선택함

Estimate Parameters

In [ ]:
```
# 참고 #
train_data.values
```

In [16]:
```
# ARIMA model fitting
# The (p, d, q) order of the model for the number of AR parameters, differences, and MA parameters to use.

model = ARIMA(train_data.values, order=(1,1,0))
model_fit = model.fit()
model_fit.summary()

# AIC 값은 1069.44이고, constant의 p-value 값이 유의미하지 않게 나왔다.
```

Out[16]:

ARIMA Model Results

Dep. Variable:	D.y	**No. Observations:**	114
Model:	ARIMA(1, 1, 0)	**Log Likelihood**	-531.720
Method:	css-mle	**S.D. of innovations**	25.659
Date:	Wed, 09 Dec 2020	**AIC**	1069.440
Time:	22:26:11	**BIC**	1077.649
Sample:	1	**HQIC**	1072.771

	coef	std err	z	P>|z|	[0.025	0.975]
const	3.5124	3.329	1.055	0.291	-3.012	10.037
ar.L1.D.y	0.2803	0.091	3.077	0.002	0.102	0.459

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	3.5681	+0.0000j	3.5681	0.0000

Diagnosis Check - ARIMA

In [17]:
```
# Parameter search

print('Examples of parameter combinations for Seasonal ARIMA...')
p = range(0,3)
d = range(1,2)
q = range(0,3)
```

```
Examples of parameter combinations for Seasonal ARIMA...
```

In [18]:
```
pdq = list(itertools.product(p, d, q))
pdq
```

Out[18]:
```
[(0, 1, 0),
 (0, 1, 1),
 (0, 1, 2),
 (1, 1, 0),
 (1, 1, 1),
 (1, 1, 2),
 (2, 1, 0),
 (2, 1, 1),
 (2, 1, 2)]
```

In [19]:
```
aic=[]
for i in pdq:
    model = ARIMA(train_data.values, order=(i))
    model_fit = model.fit()
    print(f'ARIMA: {i} >> AIC : {round(model_fit.aic, 2)}')
    aic.append(round(model_fit.aic,2))
```

```
ARIMA: (0, 1, 0) >> AIC : 1076.52
ARIMA: (0, 1, 1) >> AIC : 1064.62
ARIMA: (0, 1, 2) >> AIC : 1061.08
ARIMA: (1, 1, 0) >> AIC : 1069.44
ARIMA: (1, 1, 1) >> AIC : 1058.83
ARIMA: (1, 1, 2) >> AIC : 1046.05
ARIMA: (2, 1, 0) >> AIC : 1066.2
ARIMA: (2, 1, 1) >> AIC : 1045.66
ARIMA: (2, 1, 2) >> AIC : 1047.19
```

In [20]:
```
# Search optimal parameters

optimal = [(pdq[i], j) for i, j in enumerate(aic) if j == min(aic)]
optimal
```

Out[20]:
```
[((2, 1, 1), 1045.66)]
```

In [21]:
```
# 위 최적 값으로 만든 모델 다시 Summary

model_opt = ARIMA(train_data.values, order = optimal[0][0])
model_opt_fit = model_opt.fit()
model_opt_fit.summary()

# AIC score가 1045.66으로 임의의 모델보다 성능이 좋아졌고, p-value도 모두 유의미하게 나옴
```

Out[21]:

ARIMA Model Results

Dep. Variable:	D.y	**No. Observations:**	114
Model:	ARIMA(2, 1, 1)	**Log Likelihood**	-517.830
Method:	css-mle	**S.D. of innovations**	22.317
Date:	Wed, 09 Dec 2020	**AIC**	1045.660
Time:	22:26:12	**BIC**	1059.341
Sample:	1	**HQIC**	1051.212

	coef	std err	z	P>|z|	[0.025	0.975]
const	2.5600	0.163	15.754	0.000	2.241	2.878
ar.L1.D.y	1.0890	0.085	12.816	0.000	0.922	1.256
ar.L2.D.y	-0.4730	0.086	-5.528	0.000	-0.641	-0.305
ma.L1.D.y	-0.9999	0.040	-25.219	0.000	-1.078	-0.922

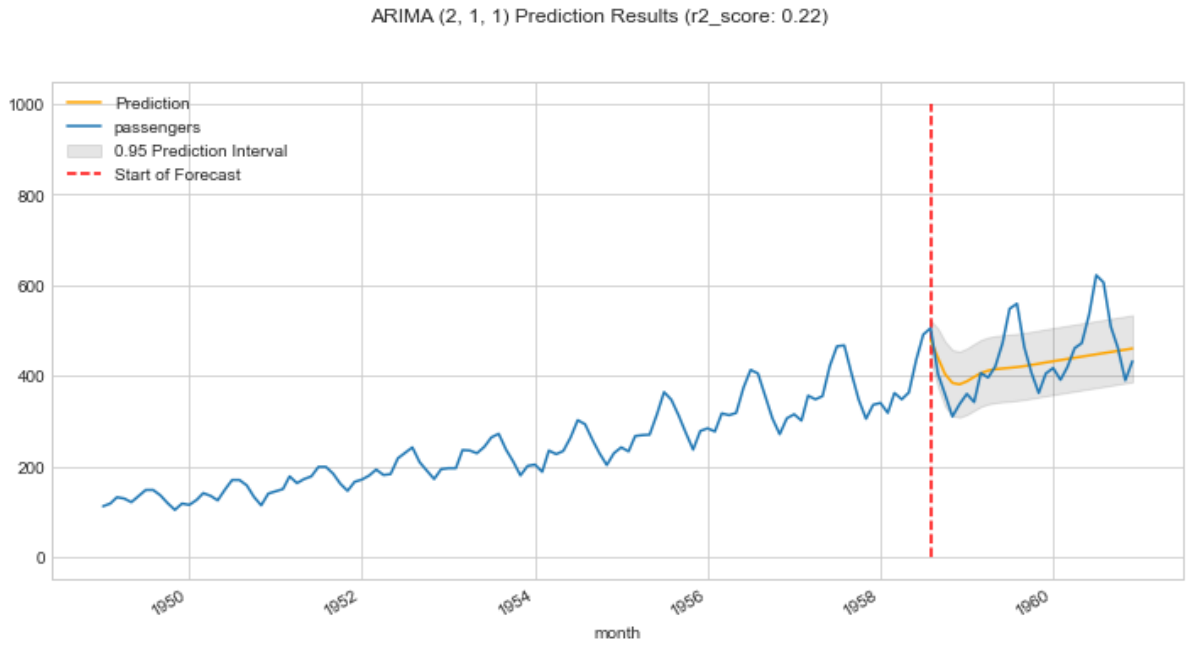
Roots

	Real	Imaginary	Modulus	Frequency
AR.1	1.1512	-0.8882j	1.4540	-0.1046
AR.2	1.1512	+0.8882j	1.4540	0.1046
MA.1	1.0001	+0.0000j	1.0001	0.0000

Use Model to Forecast - ARIMA

In [22]:
```
prediction = model_opt_fit.forecast(len(test_data))
predicted_value = prediction[0]
predicted_ub = prediction[2][:,0]
predicted_lb = prediction[2][:,1]
predict_index = list(test_data.index)
r2 = r2_score(test_data, predicted_value)
```

In [56]:
```
fig, ax = plt.subplots(figsize=(12,6))

ax.plot(predict_index, predicted_value, color = 'orange', label = 'Prediction'
) # 예측값(위 vline 이후 구간에 표시됨)
ax.fill_between(predict_index, predicted_lb, predicted_ub, color = 'k', alpha
= 0.1, label = '0.95 Prediction Interval')

data.plot(ax = ax);
ax.vlines('1958-08-01', 0, 1000, linestyle = '--', color = 'r', label = 'Start
of Forecast') # x좌표를 날짜로 적음
ax.legend(loc='upper left')
plt.suptitle(f'ARIMA {optimal[0][0]} Prediction Results (r2_score: {round(r2,
2)})')

plt.show()

# 빨간 점선 이후의 주황색 선이 예측값이며, 회색 구간이 95% interval 구간이다.
# 대체로 추세를 따라가나 피크 값을 완벽히 예측하기에는 다소 무리가 있는 것을 볼 수 있다.
# R2 score도 0.22 수준인 것을 확인할 수 있었다.
```



(참고) Diagnosis Sheck - SARIMA

- SARIMA까지 해야하는 것인지는 모르겠음

In [58]:
```
print('Examples of parameter combinations for Seasonal ARIMA...')
p = range(0,3)
d = range(1,2)
q = range(0,3)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12 # 계절성이 12개월마다 있다고 생각해서 12 입력. 얘도 위에 range로 탐색해도 됨
                ) for x in list(itertools.product(p, d, q))]
seasonal_pdq
```

```
Examples of parameter combinations for Seasonal ARIMA...
```

Out[58]:
```
[(0, 1, 0, 12),
 (0, 1, 1, 12),
 (0, 1, 2, 12),
 (1, 1, 0, 12),
 (1, 1, 1, 12),
 (1, 1, 2, 12),
 (2, 1, 0, 12),
 (2, 1, 1, 12),
 (2, 1, 2, 12)]
```

In [ ]:
```
aic = []
params = []
for i in pdq:
    for j in seasonal_pdq:
        try:
            model = SARIMAX(train_data.values, order=(i), seasonal_order = (j))
            model_fit = model.fit()
            print(f'SARIMA: {i}{j} >> AIC : {round(model_fit.aic,2)}')
            aic.append(round(model_fit.aic,2))
            params.append((i, j))
        except:
            continue
```

In [69]:
```
# Search optimal parameters

optimal = [(params[i], j) for i, j in enumerate(aic) if j == min(aic)]
optimal

# small pdq는 (1,1,0), large pdq는 (1,1,2) 그리고 Seasonal parameter는 12인 것을 볼 수 있다.
```

Out[69]:
```
[(((1, 1, 0), (1, 1, 2, 12)), 751.15)]
```

In [70]:
```
model_opt = SARIMAX(train_data.values, order=optimal[0][0][0], seasonal_order
= optimal[0][0][1])
model_opt_fit = model_opt.fit()
model_opt_fit.summary()

# ARIMA보다 SARIMA가 AIC가 훨씬 낮은 것을 볼 수 있다.
```

Out[70]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	115
Model:	SARIMAX(1, 1, 0)x(1, 1, [1, 2], 12)	**Log Likelihood**	-370.575
Date:	Wed, 02 Dec 2020	**AIC**	751.150
Time:	23:07:23	**BIC**	764.274
Sample:	0	**HQIC**	756.464
	- 115		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.2362	0.093	-2.541	0.011	-0.418	-0.054
ar.S.L12	0.9981	0.192	5.208	0.000	0.622	1.374
ma.S.L12	-1.3630	2.095	-0.651	0.515	-5.470	2.744
ma.S.L24	0.3914	0.731	0.536	0.592	-1.041	1.823
sigma2	70.2975	134.586	0.522	0.601	-193.487	334.082

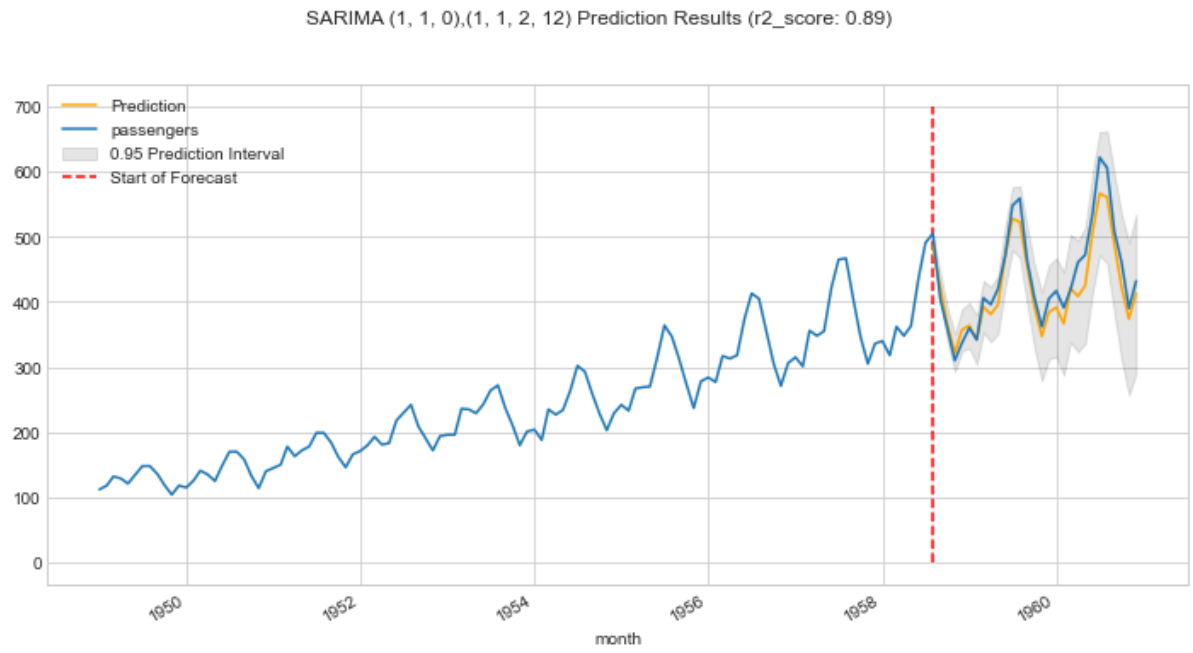
Ljung-Box (Q):	39.85	**Jarque-Bera (JB):**	2.76
Prob(Q):	0.48	**Prob(JB):**	0.25
Heteroskedasticity (H):	1.09	**Skew:**	0.38
Prob(H) (two-sided):	0.79	**Kurtosis:**	2.72

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Use Model to Forecast - SARIMA

In [72]:
```
prediction = model_opt_fit.get_forecast(len(test_data))
predicted_value = prediction.predicted_mean
predicted_ub = prediction.conf_int()[:,0]
predicted_lb = prediction.conf_int()[:,1]
predict_index = list(test_data.index)
r2 = r2_score(test_data, predicted_value)
```

In [74]:
```
fig, ax = plt.subplots(figsize=(12,6))

ax.plot(predict_index, predicted_value, color = 'orange', label = 'Prediction'
) # 예측값(위 vline 이후 구간에 표시됨)
ax.fill_between(predict_index, predicted_lb, predicted_ub, color = 'k', alpha
= 0.1, label = '0.95 Prediction Interval')

data.plot(ax = ax);
ax.vlines('1958-08-01', 0, 700, linestyle = '--', color = 'r', label = 'Start
 of Forecast') # x좌표를 날짜로 적음
ax.legend(loc='upper left')
plt.suptitle(f'SARIMA {optimal[0][0][0]},{optimal[0][0][1]} Prediction Results
(r2_score: {round(r2,2)})')

plt.show()

# 예측 값의 추세가 실제 값을 상당히 잘 따라가고 있으며,
# r2 score가 0.89로 훨씬 더 성능이 향상됨
# 계절성을 반영한 것이 예측 성능을 향상시키는데 기여를 했다고 볼 수 있다.
```



Diagnosis Check - auto_arima (ADP에서는 패키지 없을 듯)

In [ ]:
```
# Parameters search

auto_arima_model = auto_arima(train_data, start_p=1, start_q=1, # p, q 시작값
                              max_p=3, max_q=3, # p, q 최대값
                              m=12, seasonal=True, # 계절성 있는지와 구간(True = SARIMA) / 계절성 없다고 생각되면 m빼고, False
                              d=1, D=1, # 차분 최소, 최대
                              max_P=3, max_Q=3, # P, Q 최대  ## start_P, start_Q의 default 값은 1이라 생략했음
                              trace=True,  # 각 결과값은 print 해줌
                              error_action='ignore',
                              suppress_warnings=True,
                              stepwise=False)
```

In [76]:
```
auto_arima_model.summary()
# AIC가 751
```

Out[76]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	115
Model:	SARIMAX(1, 1, 0)x(1, 1, [1, 2], 12)	**Log Likelihood**	-370.575
Date:	Wed, 02 Dec 2020	**AIC**	751.150
Time:	23:14:54	**BIC**	764.274
Sample:	0	**HQIC**	756.464
	- 115		
Covariance Type:	opg		

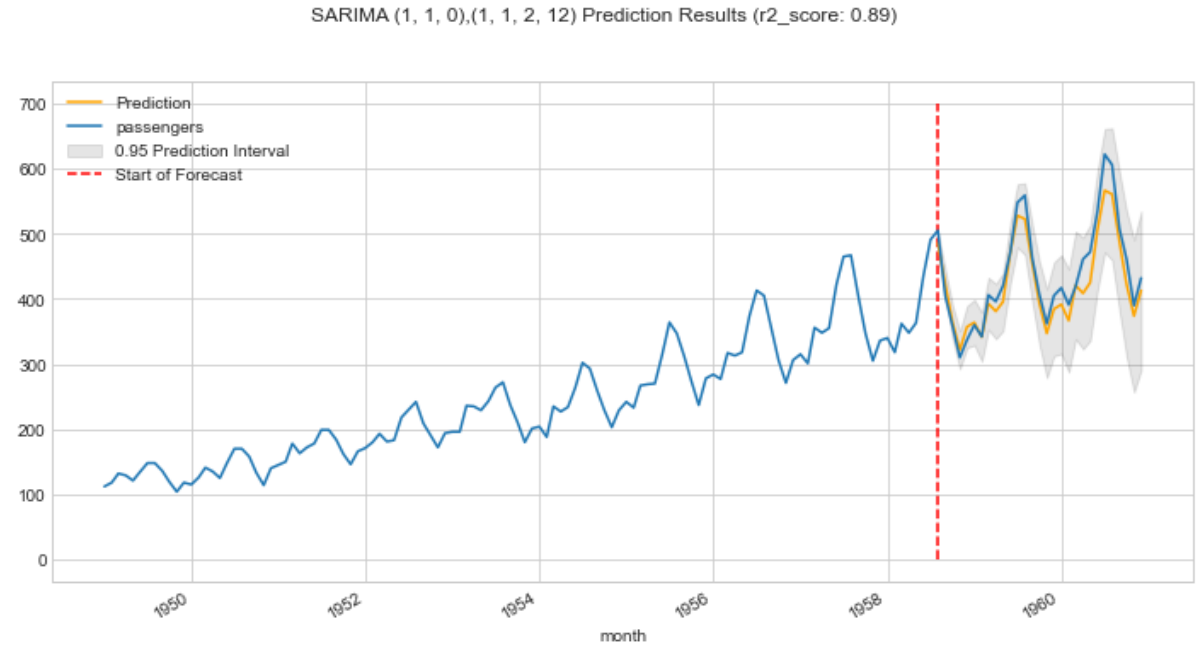
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.2362	0.093	-2.541	0.011	-0.418	-0.054
ar.S.L12	0.9981	0.192	5.208	0.000	0.622	1.374
ma.S.L12	-1.3630	2.095	-0.651	0.515	-5.470	2.744
ma.S.L24	0.3914	0.731	0.536	0.592	-1.041	1.823
sigma2	70.2975	134.586	0.522	0.601	-193.487	334.082

Ljung-Box (Q):	39.85	**Jarque-Bera (JB):**	2.76
Prob(Q):	0.48	**Prob(JB):**	0.25
Heteroskedasticity (H):	1.09	**Skew:**	0.38
Prob(H) (two-sided):	0.79	**Kurtosis:**	2.72

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [77]:
```
prediction = auto_arima_model.predict(len(test_data), return_conf_int=True)
predicted_value = prediction[0]
predicted_ub = prediction[1][:,0]
predicted_lb = prediction[1][:,1]
predict_index = list(test_data.index)
r2 = r2_score(test_data, predicted_value)
```

In [78]:
```
fig, ax = plt.subplots(figsize=(12,6))

ax.plot(predict_index, predicted_value, color = 'orange', label = 'Prediction'
) # 예측값(위 vline 이후 구간에 표시됨)
ax.fill_between(predict_index, predicted_lb, predicted_ub, color = 'k', alpha
= 0.1, label = '0.95 Prediction Interval')

data.plot(ax = ax);
ax.vlines('1958-08-01', 0, 700, linestyle = '--', color = 'r', label = 'Start
 of Forecast') # x좌표를 날짜로 적음
ax.legend(loc='upper left')
plt.suptitle(f'SARIMA {optimal[0][0][0]},{optimal[0][0][1]} Prediction Results
(r2_score: {round(r2,2)})')

plt.show()

# 예측 값의 추세가 실제 값을 상당히 잘 따라가고 있으며,
# r2 score가 0.89로 훨씬 더 성능이 향상됨
# 계절성을 반영한 것이 예측 성능을 향상시키는데 기여를 했다고 볼 수 있다.
```



요인분석

- 기본적으로는 상관관계가 높은 변수끼리 그룹핑하는 것이기에 변수간의 상관관계가 전반적으로 매우 낮다면 요인분석에 부적합한 자료라고 볼 수 있다

- <요인분석의 용도>
- ① 데이터의 양을 줄여 정보를 요약하는 경우
- ② 변수들 내부에 존재하는 구조를 파악하려는 경우
- ③ 요인으로 묶어지지 않는 변수 중 중요도가 낮은 변수를 제거하고자 하는 경우
- ④ 같은 개념을 측정하려고 하는 변수들이 동일한 요인으로 묶이는지 확인하고자 하는 경우
- ⑤ 요인분석을 통하여 얻어진 요인들을 회귀분석이나 판별분석에서 설명변수로 활용하고자 하는 경우

- 요인분석에서 요인수를 결정하는 방법은 크게 3가지가 있다.
- 1. 고유값(eigen value)
- 한 요인에 대한 '요인적적재량 제곱의 합'
- 고유값이 크다는 것은 요인이 변수들의 분산을 잘 설며명한다는 의미
- 고유값을 기준으로 할 때는 보통 1 이상을 갖는 요인 수 추출
- 1. 총 설명력(total cumulative)
- 절대적인 기준은 없으나 사회과학에서는 대체적으로 60% 내외로 결정
- 1. 스크리토표(screen table)
- 도표의 감소폭이 체감하기 직전까지의 요인 수를 기준으로 추출

- 방법 : 직각회전과 사각회전으로 나뉜다.
- 직각회전은 요인 간의 독립성을 유지하여 요인행렬이 뚜렷해질때까지 회전한다. 종류는 varimax, quartimax, equimax
- 사각회전은 요인 간의 독립성을 유지하지 않고 뚜렷해질때까지 회전시킨다. 종류는 oblimin, Promax

In [194]:
```
import pandas as pd
df = pd.read_csv('./data/Airline_Passenger_satisfaction_train.csv')
df = df.drop(columns=['Unnamed: 0'])
```

In [195]:
```
df.head()
```

Out[195]:

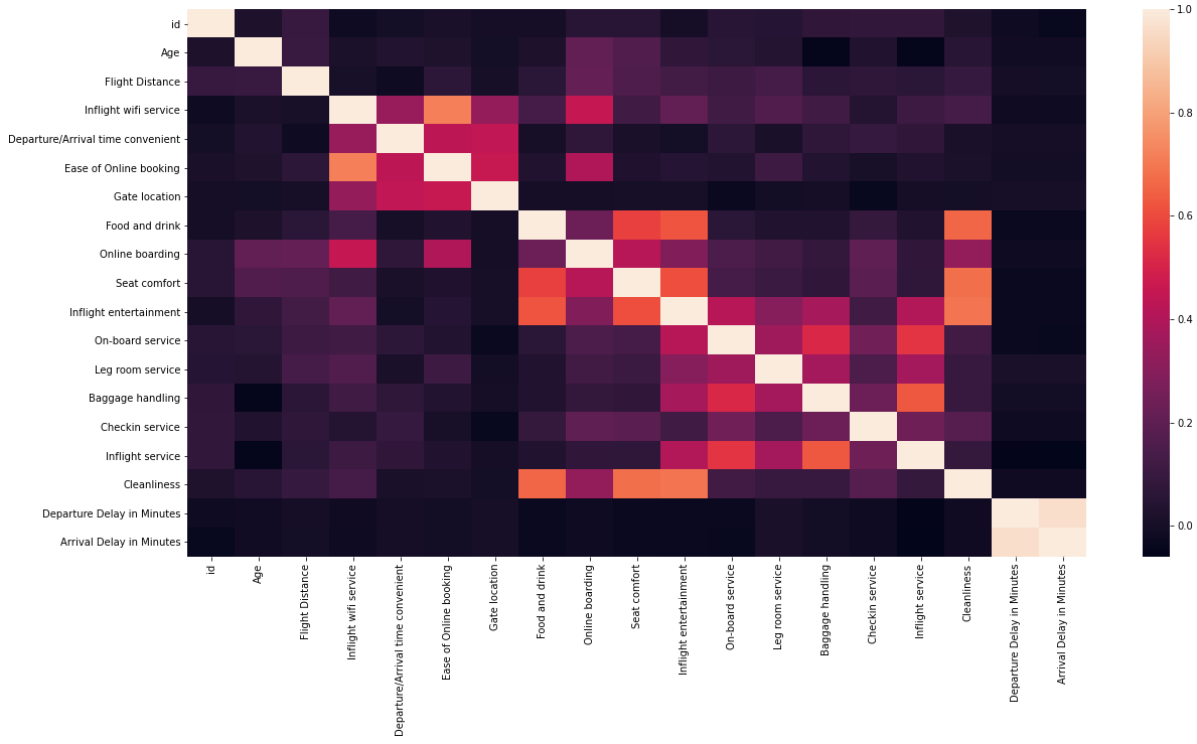
	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ea O boo
0	70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	4	
1	5047	Male	disloyal Customer	25	Business travel	Business	235	3	2	
2	110028	Female	Loyal Customer	26	Business travel	Business	1142	2	2	
3	24026	Female	Loyal Customer	25	Business travel	Business	562	2	5	
4	119299	Male	Loyal Customer	61	Business travel	Business	214	3	3	

5 rows × 24 columns

- 요인 분석의 첫 번째 단계는 모든 변수의 상관 관계도를보고 어떤 변수가 쓸모 없거나 다른 변수와 너무 상관 관계가 있는지 확인하는 것입니다.

In [196]:
```
import seaborn as sns
plt.figure(figsize=(20,10))
c= df.corr()
sns.heatmap(c)
```

Out[196]: <AxesSubplot:>



- 일부 변수, 특히 설문 조사 답변과 관련된 변수는 매우 높은 상관 관계를 보입니다. 그러나 실제로 눈에 띄는 것은 "출발 지연 (분)"과 "도착 지연 (분)"사이에 매우 높은 상관 관계 (0.98)입니다. 말이 되네요. 비행기가 예상보다 늦게 출발하면 늦게 도착해야합니다. 이 매우 높은 상관 관계를 고려하여 데이터 세트에서 해당 열을 제거하기로 결정했습니다.

In [ ]:
```
# df.drop(['Arrival Delay in Minutes'], axis=1, inplace=True)
```

- 이제 쓸모없는 변수를 제거 했으므로 요인 분석에 사용할 14 개의 변수가 있습니다.

In [177]:
```
x = df[[
'Inflight wifi service',
'Departure/Arrival time convenient',
'Ease of Online booking',
'Gate location',
'Food and drink',
'Online boarding',
'Seat comfort',
'Inflight entertainment',
'On-board service',
'Leg room service',
'Baggage handling',
'Checkin service',
'Inflight service',
'Cleanliness']]
```

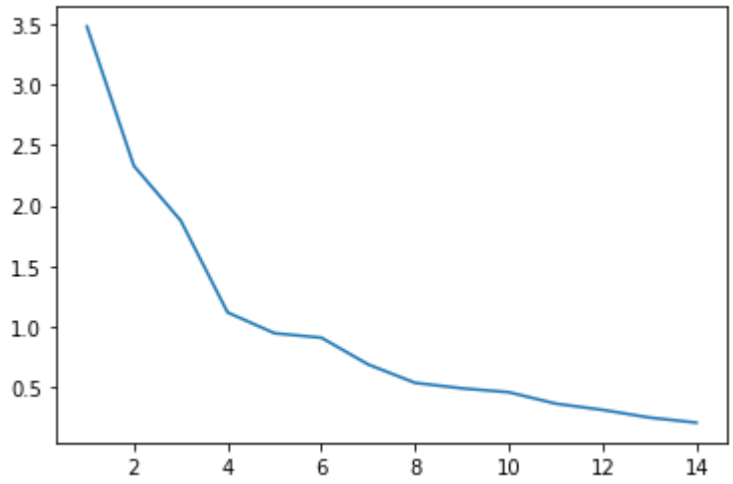
- Factor Analysis
- 14 개의 변수를 더 적은 수의 잠재 변수(요인)로 설명 할 수 있는지 확인해본다.

- 얼마나 많은 요인이 필요한지 알아 내기 위해 고유 값(eigenvalues)을 볼 수 있습니다. 고유 값은 요인이 설명하는 변수의 분산 정도를 측정하는 것입니다. 고유 값이 1보다 크면 요인이 고유 변수보다 더 많은 분산을 설명한다는 것을 의미합니다. 고유 값 2.5는 요인이 2.5 변수의 분산을 설명한다는 것을 의미합니다.

In [199]:
```
#Subset of the data, the 14 columns containing the survey answers
x =df[df.columns[6:20]]
fa = FactorAnalyzer()
fa.fit(x, 10)

#Get Eigen values and plot them
ev, v = fa.get_eigenvalues()
print(ev)
plt.plot(range(1,x.shape[1]+1),ev)
```

```
[3.47460753 2.32753739 1.87688179 1.11982655 0.94867075 0.911874
 0.69120616 0.53988526 0.49477839 0.46265754 0.36863288 0.31738585
 0.25404277 0.21201313]
```

Out[199]:
```
[<matplotlib.lines.Line2D at 0x1e44ff0a5c8>]
```



- 세 번째 요소 이후 고유 값의 큰 감소를 고려할 때 여기서는 세 가지 요소 만 사용합니다. 이러한 요인은 3.5, 2.3 및 1.9의 고유 값을 가지며 이는 약 7.7 변수의 분산을 설명 함을 의미합니다.
- FactorAnalyzer 함수는 우리가 원하는 요소의 수와 회전 유형을 지정하는 곳입니다. 간단히 말해서, 회전의 개념은 더 간단하고 해석 가능한 구조를 얻기 위해 요소를 회전하는 것입니다. 많은 유형의 회전이 존재합니다. 아래에서는 생성 된 요인이 상관 (직교성)되지 않도록 보장하면서 제곱 하중 분산의 합을 최대화하는 varimax 회전을 사용합니다. 어떤 요소가 생성되는지 살펴 보겠습니다.

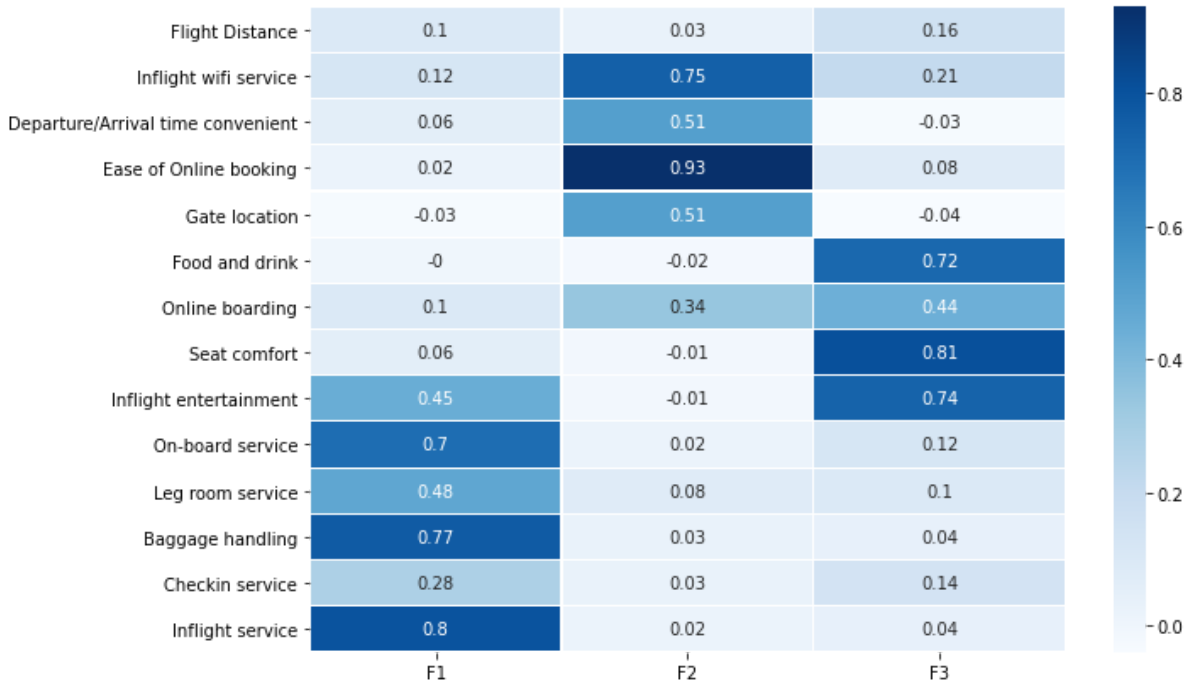
In [209]:
```
fa = FactorAnalyzer(3, rotation='varimax')
fa.fit(x)
loads = pd.DataFrame(fa.loadings_, index = x.columns,
                     columns = ['F1','F2','F3'])
# print(loads)
loads
```

Out[209]:

	F1	F2	F3
Flight Distance	0.100116	0.032710	0.162905
Inflight wifi service	0.117115	0.747520	0.209846
Departure/Arrival time convenient	0.064318	0.508389	-0.033294
Ease of Online booking	0.016619	0.927371	0.075719
Gate location	-0.027473	0.510436	-0.035619
Food and drink	-0.002526	-0.020400	0.722337
Online boarding	0.099692	0.342982	0.442006
Seat comfort	0.057283	-0.007236	0.812632
Inflight entertainment	0.445210	-0.014473	0.739131
On-board service	0.696665	0.024657	0.124992
Leg room service	0.479079	0.077032	0.102332
Baggage handling	0.765447	0.028737	0.038456
Checkin service	0.282889	0.025488	0.142322
Inflight service	0.799465	0.019260	0.035335

In [216]:
```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10,7))
sns.heatmap(loads.round(2), annot=True, cmap='Blues', linewidths=.1)
```

Out[216]: <AxesSubplot:>



- 요인 loadings 수치가 높을수록 해당 요인에 대한 변수가 더 중요합니다. 여기서 0.5의 로딩 컷오프가 사용됩니다. 이 컷오프는 어떤 변수가 어떤 요인에 속하는지 결정합니다. 예를 들어, 첫 번째 요인에는 변수 5, 7, 8 및 14 (각각 0.75, 0.78, 0.74 및 0.85의 로딩)가 포함되어 있습니다.

- 생성 된 3 가지 요인, 포함 된 변수 및 가능한 "해석 성"은 다음과 같습니다.
- 편안함 : 식음료, 편안한 좌석, 기내 엔터테인먼트, 청결도
- 서비스 : 기내 서비스, 수하물 처리, 기내 서비스
- 편의 : 기내 와이파이, 출발 / 도착 시간 편의, 온라인 예약, 게이트 위치.
- 자, 훌륭하지만 우리의 요인이 좋은지 어떻게 알 수 있습니까? 음, Cronbach 알파는 요인의 변수가 "일관되고"신뢰할 수있는 요인을 형성하는지 여부를 측정하는 데 사용할 수 있습니다. 알파에 대해 0.6 이상의 값은 실제로 허용되는 것으로 간주됩니다. 다음은 pingouin 패키지를 사용하여 Cronbach 알파를 얻는 코드입니다.

In [211]:
```
import pingouin as pg
#Create the factors
factor1 = df[['Food and drink', 'Seat comfort', 'Inflight entertainment', 'Cle
anliness']]
factor2 = df[['On-board service', 'Baggage handling', 'Inflight service']]
factor3 = df[['Inflight wifi service', 'Departure/Arrival time convenient', 'E
ase of Online booking', 'Gate location']]
#Get cronbach alpha
factor1_alpha = pg.cronbach_alpha(factor1)
factor2_alpha = pg.cronbach_alpha(factor2)
factor3_alpha = pg.cronbach_alpha(factor3)
print(factor1_alpha, factor2_alpha, factor3_alpha)
```

```
(0.87628779166241, array([0.875, 0.878])) (0.7942916933090223, array([0.792,
0.796])) (0.7679754211110685, array([0.766, 0.77 ]))
```

In [224]:
```
# 직접 계산해주는 방법
def cronbach_alpha(x):
    xdim=x.shape
    cvsum=x.var(axis=0).sum()
    rsvar=x.sum(axis=1).var()
    ca=xdim[1]/(xdim[1]-1)*(1-cvsum/rsvar)
    return (ca)

cronbach_alpha(factor1)
```

Out[224]:
```
0.8762877916624833
```

- 알파는 0.87, 0.79 및 0.76에서 평가되며 이는 유용하고 일관성이 있음을 나타냅니다. 이러한 새로운 요인을 다른 분석이나 예측을위한 변수로 사용할 수 있습니다. 다음은 전체 데이터 프레임에 요인을 적용하고 14 개의 변수를 대체 할 수있는 3 개의 새 변수를 만드는 코드입니다.

In [215]:
```
new_variables = fa.fit_transform(x)
pd.DataFrame(new_variables, columns = ['F1','F2','F3'])
```

Out[215]:

	F1	F2	F3
0	0.593968	0.022783	1.148622
1	-0.422688	0.192866	-1.836159
2	0.308417	-0.761049	1.354437
3	-0.327896	1.327206	-1.313135
4	-0.340515	0.133981	0.626539
...	...	...	...
103899	-0.412505	-0.508426	-1.117300
103900	1.439517	0.905254	0.637774
103901	0.350340	-1.248267	0.557497
103902	-0.232795	-0.984242	-1.933920
103903	-0.665903	0.086078	-1.886736

103904 rows × 3 columns

PCA(주성분분석)

In [225]:
```
import pandas as pd
```

In [233]:
```
df = pd.read_csv('./data/iris.csv', header=0, names=['Unnamed: 0', 'sepal length', 'sepal width',
                                                     'petal length', 'petal width', 'target'])
.drop(columns=['Unnamed: 0'])
df.head()
```

Out[233]:

	sepal length	sepal width	petal length	petal width	target
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

- PCA는 정규화가 필수다

In [234]:
```
from sklearn.preprocessing import StandardScaler
features = ['sepal length', 'sepal width', 'petal length', 'petal width']

# Separating out the features
x = df.loc[:, features].values

# Separating out the target
y = df.loc[:,['target']].values

# Standardizing the features
x = StandardScaler().fit_transform(x)
```

- PCA 수행
- component를 2 개로 실행해본다

In [258]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2', 'p
rincipal component 3'])
principalDf.head()
```

Out[258]:

	principal component 1	principal component 2	principal component 3
0	-2.264703	0.480027	-0.127706
1	-2.080961	-0.674134	-0.234609
2	-2.364229	-0.341908	0.044201
3	-2.299384	-0.597395	0.091290
4	-2.389842	0.646835	0.015738

In [259]:
```
finalDf = pd.concat([principalDf, df[['target']]], axis = 1)
finalDf.head()
```

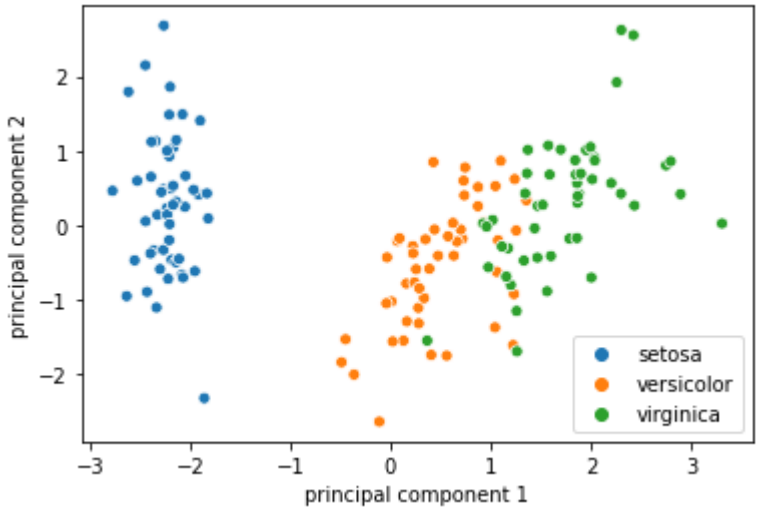
Out[259]:

	principal component 1	principal component 2	principal component 3	target
0	-2.264703	0.480027	-0.127706	setosa
1	-2.080961	-0.674134	-0.234609	setosa
2	-2.364229	-0.341908	0.044201	setosa
3	-2.299384	-0.597395	0.091290	setosa
4	-2.389842	0.646835	0.015738	setosa

- 시각화

In [260]:
```
sns.scatterplot(data = finalDf, x = 'principal component 1', y = 'principal co
mponent 2',
                hue=finalDf['target'].tolist())
```

Out[260]:
```
<AxesSubplot:xlabel='principal component 1', ylabel='principal component 2'>
```



- 분산 설명
- 각 principal component별로 얼마의 정보(분산)을 설명하는지를 나타낸다.
- 일부 정보의 손실을 있다.
- 그러나 component 1로 72.96% 정보가 설명이 되고, component 2로 22.85%가 설명이 되기 때문에 두 개의 component만 사용해도 95.81%의 분산을 설명할 수 있게 된다.
- 반면, component 3의경우 3.67%의 설명력만을 가지므로 크게 영향력이 없다.

In [261]:
```
pca.explained_variance_ratio_
```

Out[261]:
```
array([0.72962445, 0.22850762, 0.03668922])
```

코호트 분석

In [9]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl

mpl.rcParams['lines.linewidth'] = 2

%matplotlib inline
```

In [19]:
```
df = pd.read_excel("./data/relay-foods.xlsx", sheet_name = 'Purchase') # Purch
ase
df.head()
```

Out[19]:

	OrderId	OrderDate	UserId	TotalCharges	CommonId	PupId	PickupDate
0	262	2009-01-11	47	50.67	TRQKD	2	2009-01-12
1	278	2009-01-20	47	26.60	4HH2S	3	2009-01-20
2	294	2009-02-03	47	38.71	3TRDC	2	2009-02-04
3	301	2009-02-06	47	53.38	NGAZJ	2	2009-02-09
4	302	2009-02-06	47	14.28	FFYHD	2	2009-02-09

- 분석 주제
- 첫 번째 구매행동 이후 몇 개월까지 구매행위가 지속되는가?
- 구매주기는 대략적으로 얼마큼 되는가?
- 첫 구매 날짜(브랜드 인입 시기)에 따른 최근 구매 패턴(구매, 비구매) 비교 등

- 상기 분석 목표에 따라서 변수를 설정하여야 하는데, 여기에서는 '첫구매'일로 설정하겠습니다.
- (가입일이 있다면 보통 가입일에 따라서 설정하기도 합니다.) ex. 가입일 후 첫 구매, 재구매, N번째 구매일

- 구매일(OrderDate)을 YYYY-MM 형식으로 만들어줌

In [20]:
```
df['OrderPeriod'] = df.OrderDate.apply(lambda x: x.strftime('%Y-%m'))
df.head()
```

Out[20]:

	OrderId	OrderDate	UserId	TotalCharges	CommonId	PupId	PickupDate	OrderPeriod
0	262	2009-01-11	47	50.67	TRQKD	2	2009-01-12	2009-01
1	278	2009-01-20	47	26.60	4HH2S	3	2009-01-20	2009-01
2	294	2009-02-03	47	38.71	3TRDC	2	2009-02-04	2009-02
3	301	2009-02-06	47	53.38	NGAZJ	2	2009-02-09	2009-02
4	302	2009-02-06	47	14.28	FFYHD	2	2009-02-09	2009-02

- 각각 사용자의 첫 구매월을 추출하기위해 UserId를 index로 설정한 이후 groupby 함수를 사용하여 (기준 index level = 0) 'CohortGroup' 변수를 추가합니다.

In [24]:
```
df = df.set_index('UserId')

# 고객 각각의 첫 구매기간 추출
df['CohortGroup'] = df.groupby(level=0)['OrderDate'].min().apply(lambda x: x.s
trftime('%Y-%m'))
df = df.reset_index()
df.head()
```

Out[24]:

	UserId	OrderId	OrderDate	TotalCharges	CommonId	PupId	PickupDate	OrderPeriod	Cohor
0	47	262	2009-01-11	50.67	TRQKD	2	2009-01-12	2009-01	
1	47	278	2009-01-20	26.60	4HH2S	3	2009-01-20	2009-01	
2	47	294	2009-02-03	38.71	3TRDC	2	2009-02-04	2009-02	
3	47	301	2009-02-06	53.38	NGAZJ	2	2009-02-09	2009-02	
4	47	302	2009-02-06	14.28	FFYHD	2	2009-02-09	2009-02	

- 첫구매일(년월)와 구매 날짜(년월)를 기준으로 하여 고객 수, 주문 수, 총매출 합계를 계산합니다.

In [27]:
```
# CohortGroup & OrderPeriod
grouped = df.groupby(['CohortGroup', 'OrderPeriod'])

cohorts = grouped.agg({'UserId': pd.Series.nunique, # DISTINCT COUNT
                       'OrderId': pd.Series.nunique,
                       'TotalCharges': np.sum}) # SUM

cohorts.rename(columns={'UserId': 'TotalUsers',
                        'OrderId': 'TotalOrders'}, inplace=True)
cohorts.head()
```

Out[27]:

		TotalUsers	TotalOrders	TotalCharges
CohortGroup	**OrderPeriod**			
2009-01	**2009-01**	22	30	1850.255
	2009-02	8	25	1351.065
	2009-03	10	26	1357.360
	2009-04	9	28	1604.500
	2009-05	10	26	1575.625

- 위의 결과를 해석해보자면 2009년 1월에 첫구매한 고객 수가 22명, 주문 횟수가 30번, 총매출 합계가 1850원입니다.
- 이후 2월에는 22명 중 8명만이 구매행동을 보였으며, 3월에는 10명, 4월에는 9명만이 구매행동을 보였습니다. 이는 첫 번째 1월에 구매행동을 보인 22명에 대한 구매 유무만을 측정하여 표로 나타낸 것으로 22명의 회원이 매달 연속적인 구매를 했다고 볼 수는 없습니다.

- <년월 - 년월>의 패턴을 <년월 - 소요기간(월)>로 변환하여 보기위해 다음과 같은 함수를 만들어줍니다.

In [28]:
```
# Label the CohortPeriod for each CohortGroup
def cohort_period(df):
    df['CohortPeriod'] = np.arange(len(df)) + 1
    return df
```

In [34]:
```
cohorts = cohorts.groupby(level=0).apply(cohort_period)
cohorts.head(20)
```

Out[34]:

		TotalUsers	TotalOrders	TotalCharges	CohortPeriod
CohortGroup	**OrderPeriod**				
2009-01	**2009-01**	22	30	1850.2550	1
	2009-02	8	25	1351.0650	2
	2009-03	10	26	1357.3600	3
	2009-04	9	28	1604.5000	4
	2009-05	10	26	1575.6250	5
	2009-06	8	26	1384.8400	6
	2009-07	8	24	1750.8400	7
	2009-08	7	21	1426.5714	8
	2009-09	7	24	1964.2755	9
	2009-10	7	13	860.3292	10
	2009-11	7	21	1821.8153	11
	2009-12	8	22	2152.1165	12
	2010-01	11	25	2084.2236	13
	2010-02	7	19	2068.7771	14
	2010-03	6	12	1504.3325	15
2009-02	**2009-02**	15	15	666.3100	1
	2009-03	3	8	501.6100	2
	2009-04	5	10	968.7800	3
	2009-05	1	2	53.3600	4
	2009-06	4	9	758.5200	5

- Retention 결과를 (%) 비율로 나타내기 위해 각각 첫 구매일(년월)에 따른 회원수를 구합니다.
- 2009-01 에 첫 구매한 회원수 22명 , 2월에 첫 구매한 회원수 15명

In [37]:
```
cohorts.reset_index(inplace=True)
cohorts.set_index(['CohortGroup', 'CohortPeriod'], inplace=True)

cohort_group_size = cohorts['TotalUsers'].groupby(level=0).first()
cohort_group_size.head()
```

Out[37]:
```
CohortGroup
2009-01    22
2009-02    15
2009-03    13
2009-04    39
2009-05    50
Name: TotalUsers, dtype: int64
```

- 생성한 결과를 divide 함수를 활용해서 각 변수를 나누어줍니다.

In [44]:
```
cohorts['TotalUsers'].unstack(0).head()
```

Out[44]:

CohortGroup	2009-01	2009-02	2009-03	2009-04	2009-05	2009-06	2009-07	2009-08	2009-09	2009-10	2009-11	2009-12
CohortPeriod												
1	22.0	15.0	13.0	39.0	50.0	32.0	50.0	31.0	37.0	54.0	130.0	65.0
2	8.0	3.0	4.0	13.0	13.0	15.0	23.0	11.0	15.0	17.0	32.0	17.0
3	10.0	5.0	5.0	10.0	12.0	9.0	13.0	9.0	14.0	12.0	26.0	18.0
4	9.0	1.0	4.0	13.0	5.0	6.0	10.0	7.0	8.0	13.0	29.0	7.0
5	10.0	4.0	1.0	6.0	4.0	7.0	11.0	6.0	13.0	13.0	13.0	NaN

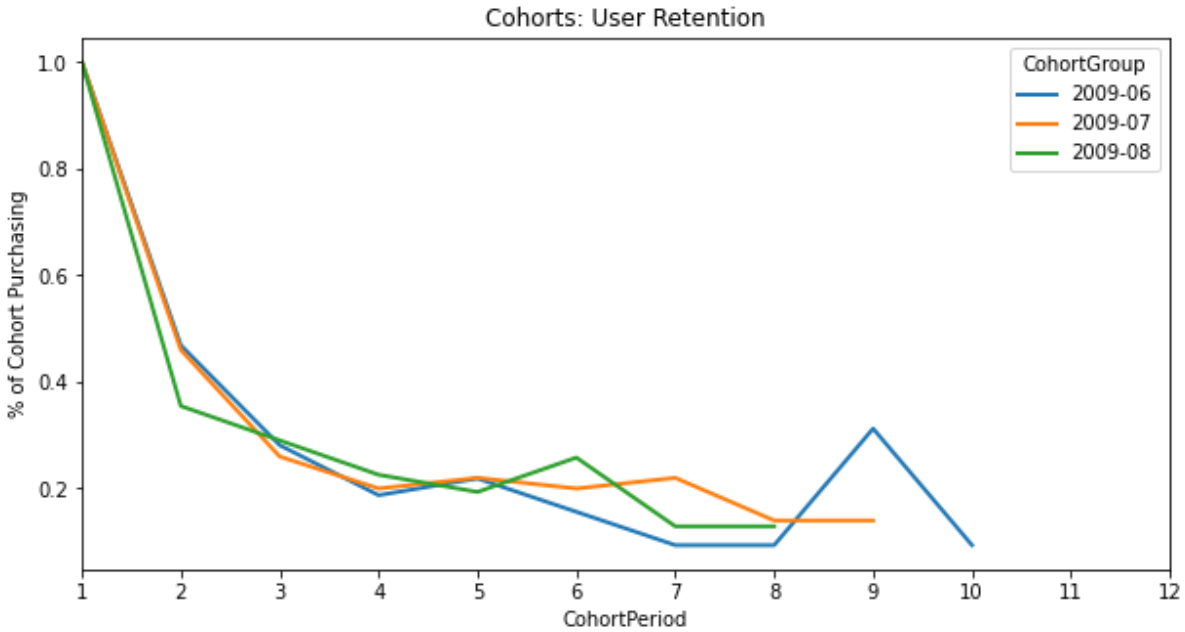
In [38]:
```
user_retention = cohorts['TotalUsers'].unstack(0).divide(cohort_group_size, axis=1)
user_retention.head(10)
```

Out[38]:

CohortGroup	2009-01	2009-02	2009-03	2009-04	2009-05	2009-06	2009-07	2009-08	2009-09
CohortPeriod									
1	1.000000	1.000000	1.000000	1.000000	1.00	1.00000	1.00	1.000000	1.000000
2	0.363636	0.200000	0.307692	0.333333	0.26	0.46875	0.46	0.354839	0.405405
3	0.454545	0.333333	0.384615	0.256410	0.24	0.28125	0.26	0.290323	0.378378
4	0.409091	0.066667	0.307692	0.333333	0.10	0.18750	0.20	0.225806	0.216216
5	0.454545	0.266667	0.076923	0.153846	0.08	0.21875	0.22	0.193548	0.351351
6	0.363636	0.266667	0.153846	0.179487	0.12	0.15625	0.20	0.258065	0.243243
7	0.363636	0.266667	0.153846	0.102564	0.06	0.09375	0.22	0.129032	0.216216
8	0.318182	0.333333	0.230769	0.153846	0.10	0.09375	0.14	0.129032	NaN
9	0.318182	0.333333	0.153846	0.051282	0.10	0.31250	0.14	NaN	NaN
10	0.318182	0.266667	0.076923	0.102564	0.08	0.09375	NaN	NaN	NaN

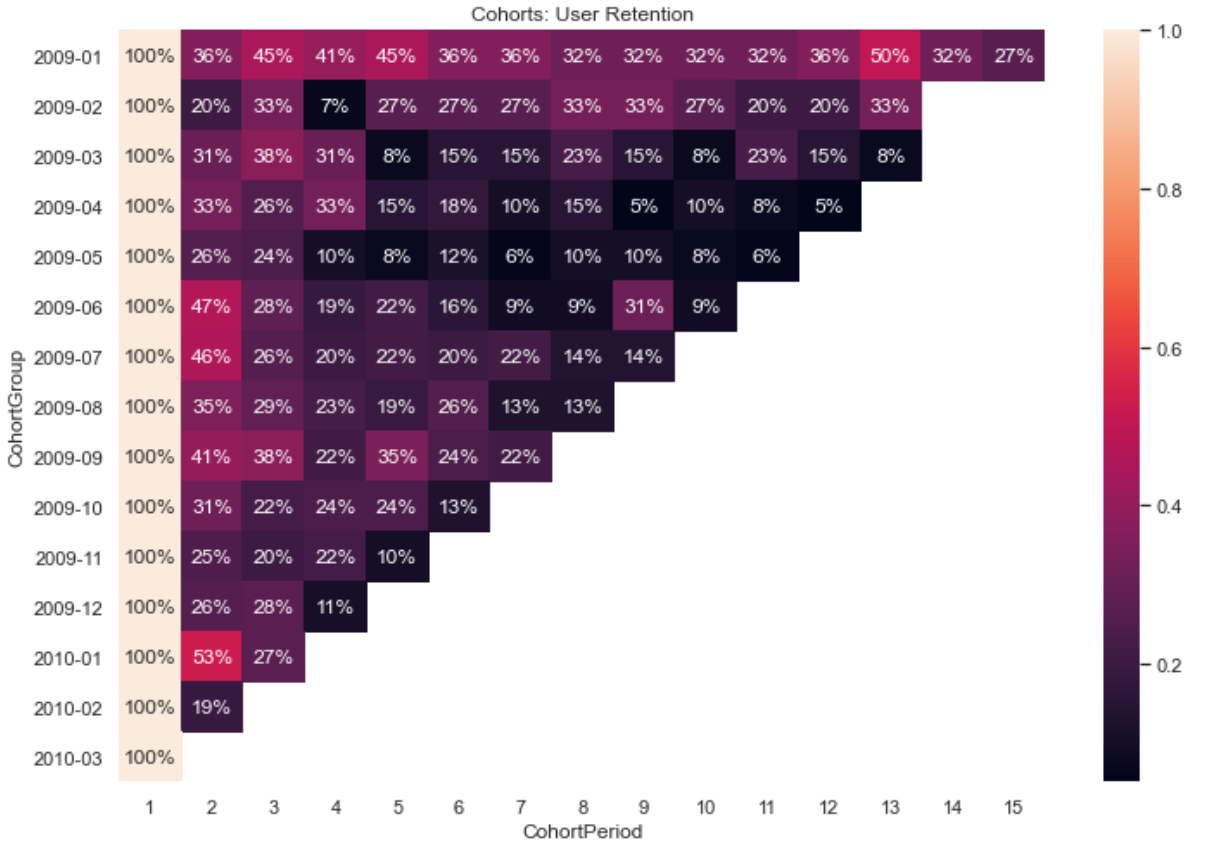
- 결과를 그래프로도 나타내 봅니다.

In [45]:
```
user_retention[['2009-06', '2009-07', '2009-08']].plot(figsize=(10,5))
plt.title('Cohorts: User Retention')
plt.xticks(np.arange(1, 12.1, 1))
plt.xlim(1, 12)
plt.ylabel('% of Cohort Purchasing');
```



- 이제 가장 일반적인 Cohort 분석의 그래프 형태로 나타내 봅니다.

In [46]:
```
import seaborn as sns
sns.set(style='white')

plt.figure(figsize=(12, 8))
plt.title('Cohorts: User Retention')
sns.heatmap(user_retention.T, mask=user_retention.T.isnull(), annot=True, fmt=
'.0%');
```



- 그래프는 각각 브랜드에 유입된 시기를 비교하여 첫구매 이후 고객들의 구매패턴을 나타낸 것입니다.
- 2009년 07월에 유입된 고객들의 패턴을 기준으로 살펴보면 첫 구매 이후 한 달 뒤 46%가 구매행동을 보였고 8개월이 지난 후에는 14%만이 구매행동을 보였습니다. 여기서 자세히 살펴보면 최근 기준으로 구매행동을 보인 고객들의 유입 날짜(년월)를 확인할 수 있습니다.

- 다음과 같이 최근 발생한 매출의 회원들의 첫 구매일 분포도 확인할 수 있습니다.

In [ ]:
```
import seaborn as sns
sns.set(style='white')

plt.figure(figsize=(24, 8))
plt.title('Cohorts: User Retention & Amount of this month')
sns.heatmap(cohorts['TotalCharges'].unstack(0).T.fillna(0).astype('int'),
            mask=user_retention.T.isnull(), annot=True, fmt= '0');
```

추천 시스템

라이브러리 로딩

In [12]:
```
import pandas as pd
from surprise import SVD
from surprise import Dataset, Reader
from surprise import accuracy
from surprise.model_selection import train_test_split
```

데이터 로딩

In [16]:
```
ratings = pd.read_csv('./data/ratings.csv')
```

In [17]:
```
ratings.head(3)
```

Out[17]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224

In [18]:
```
reader = Reader(line_format='user item rating timestamp', sep=',', rating_scal
e = (0.5, 5))
# line_format = 파일의 컬럼 구성, sep 구분자, rating_scale 0.5점 단위로 최대 5점까지
# surprise는 데이터를 사용자, 아이템, 평점 세 개 컬럼만 받기 때문에 네번째부터는 자동 삭제
됨

data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader=r
eader)
trainset, testset = train_test_split(data, test_size=.25, random_state=0)
```

알고리즘 객체 생성

- SVD로 잠재 요인 협업 필터링 : 사용자가 아직 평점을 매기지 않은 아이템에 대한 평점을 예측하여 이를 추천에 반영하는 방식

In [19]:
```
algo=SVD(n_factors = 50, random_state=0)
# n_factor : 잠재요인 크기 K값(default 100, 커질수록 정확도 높아질 수 있으나 과적합 문제 발생)
# n_epochs : SGD 수행 시 반복 횟수, 디폴트 20
# biased : 베이스라인 사용자 편향 적용 여부, 디폴트 True
# 알고리즘
# SVD : 행렬 분해를 통해 잠재 요인 협업 필터링을 위한 알고리즘. 사용자 베이스라인 편향성을 감안한 평점 예측에 Resularizationd을 적용한 것
# KNNBasic:최근접 이웃 협업 필터링을 위한 알고리즘
# BaselinOnly : 사용자 Bias와 아이템 Bias를 감안한 SGD 베이스라인 알고리즘
algo.fit(trainset)
```

Out[19]:
```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x1ca29c973c8>
```

교차검증과 하이퍼 파라미터 튜닝

In [25]:
```
from surprise.model_selection import cross_validate

cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8678  0.8685  0.8694  0.8718  0.8690  0.8693  0.0014
MAE (testset)     0.6676  0.6662  0.6691  0.6707  0.6665  0.6680  0.0017
Fit time          3.89    3.90    3.87    3.76    3.86    3.86    0.05
Test time         0.16    0.17    0.25    0.16    0.16    0.18    0.04
```

Out[25]:
```
{'test_rmse': array([0.86782698, 0.86848068, 0.86939148, 0.87178912, 0.86899463]),
 'test_mae': array([0.66764028, 0.66619018, 0.66912079, 0.67070139, 0.66654514]),
 'fit_time': (3.893256902694702,
  3.9029791355133057,
  3.8692140579223633,
  3.7583870887756348,
  3.8624868392944336),
 'test_time': (0.1568756103515625,
  0.1663968563079834,
  0.2485804557800293,
  0.15853142738342285,
  0.1551053524017334)}
```

In [29]:
```
from surprise.model_selection import GridSearchCV

param_grid = {'n_epochs' : [20, 40, 60],
              'n_factors' : [50, 100, 200]}
gs = GridSearchCV(SVD, param_grid, measures=['rmse','mae'], cv=3)
gs.fit(data)

print('최고 RMSE : ',gs.best_score['rmse'])
print(gs.best_params)
```

```
최고 RMSE :  0.8768221180687531
{'rmse': {'n_epochs': 20, 'n_factors': 50}, 'mae': {'n_epochs': 20, 'n_factor
s': 50}}
```

In [ ]:
```
algo=SVD(n_epochs = 20 , n_factors = 50, random_state=0)
algo.fit(trainset)
```

추천 수행

- Surprise에서 추천을 예측하는 메서드는 test()와 predict() 두 가지
- test()는 사용자-아이템 평점 데이터 세트 전체에 대해서 추천을 예측
- predict()는 개별 사용자와 영화에 대한 추천 평점을 반환

test()

In [20]:
```
predictions = algo.test(testset)
print('prediction type :', type(predictions), 'size:', len(predictions))
print('prediction 결과의 최초 5개 추출')
predictions[:5]
```

```
prediction type : <class 'list'> size: 25209
prediction 결과의 최초 5개 추출
```

Out[20]:
```
[Prediction(uid=63, iid=2000, r_ui=3.0, est=3.5016267817280697, details={'was
_impossible': False}),
 Prediction(uid=31, iid=788, r_ui=2.0, est=3.2840758900255937, details={'was_
impossible': False}),
 Prediction(uid=159, iid=6373, r_ui=4.0, est=2.804939396068158, details={'was
_impossible': False}),
 Prediction(uid=105, iid=81564, r_ui=3.0, est=3.9326180027723914, details={'w
as_impossible': False}),
 Prediction(uid=394, iid=480, r_ui=3.0, est=3.3135580105479114, details={'was
_impossible': False})]
```

- 위 결과에서 첫 3개의 Prediction 객체에서 uid, iid, est 속성을 추출해봄

In [22]:
```
[(pred.uid, pred.iid, pred.r_ui, pred.est) for pred in predictions[:3]]
```

Out[22]:
```
[(63, 2000, 3.0, 3.5016267817280697),
 (31, 788, 2.0, 3.2840758900255937),
 (159, 6373, 4.0, 2.804939396068158)]
```

predict()

In [23]:
```
uid = str(196)
iid = str(302)
pred = algo.predict(uid, iid)
print(pred)
```

```
user: 196        item: 302        r_ui = None   est = 3.50   {'was_impossibl
e': False}
```

성능 평가

- Surprise의 accuracy 모듈은 RMSE, MSE 등의 방법으로 정보 제공

In [24]:
```
accuracy.rmse(predictions)
accuracy.mse(predictions)
accuracy.mae(predictions)
```

```
RMSE: 0.8682
MSE: 0.7538
MAE:  0.6672
```

Out[24]:
```
0.6672308519742739
```

Train, Test를 나누지 않고 데이터 전체를 학습에 사용 후 사용자가 보지 않은 영화 추천

학습 데이터 만들기 : 그냥 사용 못함

In [35]:
```
import pandas as pd
from surprise import SVD
from surprise import Dataset, Reader
from surprise import accuracy
from surprise.model_selection import train_test_split
from surprise.dataset import DatasetAutoFolds

# 일단 전처리 필요
ratings = pd.read_csv('./data/ratings.csv') # 원본 csv 불러와서
ratings.to_csv('./data/ratings_noh.csv', index=False, header=False) # 인덱스와
 헤더 제거한 파일로 다시 저장
# csv 파일을 바로 불러오는 형식
reader = Reader(line_format='user item rating timestamp', sep=',', rating_scal
e=(0.5, 5))
data_folds = DatasetAutoFolds(ratings_file='./data/ratings_noh.csv', reader=re
ader)
trainset = data_folds.build_full_trainset() # 전체 데이터를 학습 데이터로 생성함
```

학습 수행

In [36]:
```
algo = SVD(n_epochs = 20, n_factors=50, random_state=0)
algo.fit(trainset)
```

Out[36]:
```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x1ca3180fa08>
```

특정 사용자가 보지 않은 영화 하나를 찾고, 그 영화의 평점을 예측

In [39]:
```
# 영화 정보 확인
movies = pd.read_csv('./data/movies.csv')
movies.head(3)
```

Out[39]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance

In [45]:
```
# 사용자 ID 9가 movieID 42를 봤는지 확인
movieIds = ratings[ratings['userId']==9]['movieId']
if movieIds[movieIds==42].count() == 0:
    print('안 봤음')
```

안 봤음

In [44]:
```
# 42번 영화가 뭔지 조회해보기
print(movies[movies['movieId']==42])
```

```
    movieId                    title              genres
38       42  Dead Presidents (1995)  Action|Crime|Drama
```

In [46]:
```
# user 9의 movie 42에 대한 평점 예측해보기
## 반드시 문자열이어야함

uid=str(9)
iid=str(42)
pred = algo.predict(uid, iid, verbose=True)
```

```
user: 9          item: 42         r_ui = None   est = 3.13   {'was_impossibl
e': False}
```

- 추천 예측 평점은 est=3.13이다.

특정 사용자가 보지 않은 영화 전체를 찾고, 그 영화의 평점을 예측

In [47]:
```
# 안 본 영화가 뭔지부터 찾기
def get_unseen_surprise(ratings, movies, userId): # ratings df, movies df, 사용자ID
    seen_movies = ratings[ratings['userId']==userId]['movieId'].tolist() # 사용자(userId)가 본 영화 추출
    total_movies = movies['movieId'].tolist() # 모든 영화 리스트 생성
    unseen_movies = [movie for movie in total_movies if movie not in seen_movies] # 전체 각 영화별로 사용자가 본 영화에 없으면 추가
    print('평점 매긴 영화:', len(seen_movies), '추천 대상 영화:',len(unseen_movies),
          '전체 영화 수:', len(total_movies))
    return unseen_movies
unseen_movies = get_unseen_surprise(ratings, movies, 9)
```

```
평점 매긴 영화: 46 추천 대상 영화: 9696 전체 영화 수: 9742
```

In [52]:
```
unseen_movies[:10] # 안 본 영화 확인(10개만 확인해봄)
```

Out[52]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [58]:
```
# 추천 함수
def recomm_movie_by_surprise(algo, userId, unseen_movies, top_n=10):
    predictions = [algo.predict(str(userId), str(movieId)) for movieId in unse
en_movies] # 안 본 영화에 대해 predict
    def sortkey_est(pred):
        return pred.est
    predictions.sort(key=sortkey_est, reverse=True) # est를 기준으로 정렬
    top_predictions = predictions[:top_n] # predictions 중 top_n을 추출

    # top 추천 영화 id, 예상 평점, 제목 추출
    top_movie_ids = [int(pred.iid) for pred in top_predictions]
    top_movie_rating = [pred.est for pred in top_predictions]
    top_movie_titles = movies[movies['movieId'].isin(top_movie_ids)]['title']

    top_movie_preds = [(id, title, rating) for id, title, rating in \
                       zip(top_movie_ids, top_movie_titles, top_movie_rating)]
    return top_movie_preds

unseen_movies = get_unseen_surprise(ratings, movies, 9)
top_movie_preds = recomm_movie_by_surprise(algo, 9, unseen_movies, top_n=10)

print('Top 10 추천 영화 리스트')
for top_movie in top_movie_preds:
    print(top_movie[1], ':', top_movie[2])
```

```
평점 매긴 영화: 46 추천 대상 영화: 9696 전체 영화 수: 9742
Top 10 추천 영화 리스트
Usual Suspects, The (1995) : 4.306302135700814
Star Wars: Episode IV - A New Hope (1977) : 4.281663842987387
Pulp Fiction (1994) : 4.278152632122759
Silence of the Lambs, The (1991) : 4.226073566460876
Godfather, The (1972) : 4.1918097904381995
Streetcar Named Desire, A (1951) : 4.154746591122658
Star Wars: Episode V - The Empire Strikes Back (1980) : 4.122016128534504
Star Wars: Episode VI - Return of the Jedi (1983) : 4.108009609093436
Goodfellas (1990) : 4.083464936588478
Glory (1989) : 4.07887165526957
```

In [ ]:

07_기계학습_1

다중 분류 모델 성능평가

In [4]:
```
# 다중 분류 모델 성능 평가

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
y_true = [0, 0, 0, 1, 1, 2, 2, 2]
y_pred = [0, 0, 2, 1, 2, 2, 2, 1]
print(confusion_matrix( y_true, y_pred))
target_names = ['class 0', 'class 1', 'class 2']
print(classification_report(y_true, y_pred, target_names=target_names))
```

```
[[2 0 1]
 [0 1 1]
 [0 1 2]]
              precision    recall  f1-score   support

     class 0       1.00      0.67      0.80         3
     class 1       0.50      0.50      0.50         2
     class 2       0.50      0.67      0.57         3

    accuracy                           0.62         8
   macro avg       0.67      0.61      0.62         8
weighted avg       0.69      0.62      0.64         8
```

- 해석 : confusion matrics는 세로축이 실제값(y_true), 가로축이 예측값(y_pred)
- 해석 : class 0의 precision은 1.00, recall 0.67, f1-score 0.80임
- 해석 : 분류결과의 accuracy는 0.62, 전체의 단순평균 precision은 0.67, recall 0.61, f1-score 0.62임
- macro avg: 단순평균
- weighted avg: 각 클래스에 속하는 표본의 갯수로 가중평균

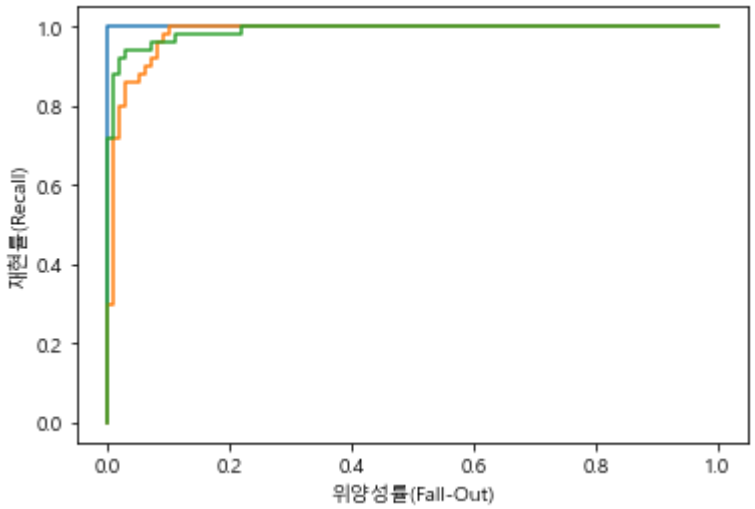
ROC 및 AUC (다중분류는 이어서 나옴)

- ROC Curve
- x축 : 허위 양성 비율(FPR) = FP/(FP+TN)
- y축 : 참 양성 비율(TPR) = TP/(FP+FN)

In [ ]:
```
model = LogisticRegression().fit(X, y)   # X, y를 fit하여 모델링
y_hat = model.predict(X)   # 예측값 생성(y_hat)

from sklearn.metrics import roc_curve

# 방법 1
fpr, tpr, thresholds = roc_curve(y, model.decision_function(X))

# 방법 2 : decision_function 메서드를 제공하지 않는 모형은 predict_proba 명령을 써서
 확률을 입력해도 된다.
fpr, tpr, thresholds = roc_curve(y, model.predict_proba(X)[:, 1])

# AUC : AUC가 1에 가까운 값이고 좋은 모형이다.
from sklearn.metrics import auc
auc(fpr1, tpr1)

# 시각화 : 모델 두 개일 때
fpr1, tpr1, thresholds1 = roc_curve(y, model1.decision_function(X)) # 모델 1
fpr2, tpr2, thresholds1 = roc_curve(y, model2.decision_function(X)) # 모델 2

plt.plot(fpr, tpr, 'o-', label="Logistic Regression") # 모델 1
plt.plot([0, 1], [0, 1], 'k--', label="random guess") # 모델 2
plt.plot([fallout], [recall], 'ro', ms=10)
plt.xlabel('위양성률(Fall-Out)')
plt.ylabel('재현률(Recall)')
plt.title('Receiver operating characteristic example')
plt.show()
```

- MultiClass Classification
- 다중 분류 문제에서는 One-versus-Rest(OvR) 문제로 가정하고 (사람, 개, 고양이를 분류하는 문제일 경우에는 사람 vs 나머지, 개 vs 나머지, 고양이 vs 나머지) 각 클래스마다 하나씩 ROC커브를 그립니다.

In [29]:
```
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.datasets import load_iris
from sklearn.preprocessing import label_binarize

iris = load_iris()
X = iris.data   # 독립변수가 있고


# 이 아래부터 활용하면 됨
y = label_binarize(iris.target, classes = [0, 1, 2])   # 종속변수 y를 더미화를 시킴

fpr = [None] * 3   # class 개수만큼(여기선 3개)
tpr = [None] * 3
threshold = [None] * 3
roc_auc = []

for i in range(3):   # range 클래스 개수만큼
    model = GaussianNB().fit(X, y[:, i])   # 모델링을 함
    fpr[i], tpr[i], threshold[i] = roc_curve(y[:, i], model.predict_proba(X)[:, 1])
    plt.plot(fpr[i], tpr[i])

    roc_auc.append(auc(fpr[i], tpr[i]))

plt.xlabel('위양성률(Fall-Out)')
plt.ylabel('재현률(Recall)')
plt.show()
print('ROC_AUC : ',roc_auc)
```




```
ROC_AUC :  [1.0, 0.9818, 0.989]
```

StandardScaler

In [ ]:
```
from sklearn.preprocessing import StandardScaler

# 스케일링 4단계
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scale = scaler.transform(X_train)
X_test_scale = scaler.transform(X_test)
```

In [ ]:
```
# 스케일링 전으로 되돌리기
scaler.inverse_transform(X_test_scale)
```

단계적 선택법 (ADP 실기 책 p.198 참고 - R 코드임)

In [ ]:
```
data = read.csv(file = 'ToyotaCorolla.csv', header=T)
data

lm_a <- lm(Price ~ ., data = data)
summary(lm_a)

result <- step(lm_a, direction="both")
result
```