DAFTAR PUSTAKA

Apriani, M. (2016). Peramalan Jumlah Pengguna Telepon Dan Estimasi Trafik Serta Analisis Parameter Jaringan Di Pt Telekomunikasi Indonesia. Politeknik Negeri Sriwijaya.

Awad, M., & Khanna, R. (2015). *Efficient Learning Machines*. Springer nature.

Cahyono, R. E., Sugiono, J. P., & Tjandra, S. (2019). Analisis Kinerja Metode *Support* Vector Regression (SVR) dalam Memprediksi Indeks Harga Konsumen. *JTIM : Jurnal Teknologi Informasi dan Multimedia*, *1*(2), 106–116. https://doi.org/10.35746/jtim.v1i2.22

Fauzy, M., Saleh W, K. R., & Asror, I. (2016). Penerapan Metode Association Rule Menggunakan Algoritma Apriori Pada Simulasi Prediksi Hujan Wilayah Kota Bandung. *Jurnal Ilmiah Teknologi Infomasi Terapan*, *2*(3). https://doi.org/10.33197/jitter.vol2.iss3.2016.111

Han, J., Kamber, M., & Pei, J. (2012). *Data Mining Concept and Techniques* (Third).

Hernu CP dan Furqon A. 2014. Pengaruh Awan Cumolonimbus terhadap Tinggi Gelombang Laut di Selat Sunda. Sekolah Tinggi Meteorologi Klimatologi dan Geofisika (STMKG). Tangerang Selatan.

Hermanto, L. A. (2018). Prakiraan Tinggi Gelombang Air Laut Menggunakan Data Mining. *Jurnal IPTEK*, *22*(1), 37. https://doi.org/10.31284/j.iptek.2018.v22i1.232

Hidayat, N. (2005). *Kajian Hidro-Oseanografi Untuk Deteksi Proses-Proses Fisik Di Pantai*. *3*(2), 13.

Hilmansyah, M. (2021, March 7). TB Syarasd I Tenggelam di Selat Makassar, Seluruh Awak Selamat. *IDN Times*. https://kaltim.idntimes.com/news/kaltim/mhilmansyah/tb-syarasd-i-tenggelam-di-selat-makassar-seluruh-awak-selamat/3

*Https://Kkp.Go.Id/*https://kkp.go.id/djprl/artikel/21045-konservasi-perairan-sebagai-upaya-menjaga-potensi-kelautan-dan-perikanan-indonesia

Ismail, H. C., & Rosana, F. C. (2022, Mei). Kapal Nelayan Tenggelam di Selat Makassar, 17 Orang Ditemukan Selamat. *Tempo.Co*. https://bisnis.tempo.co/read/1596255/kapal-nelayan-tenggelam-di-selat-makassar-17-orang-ditemukan-selamat

Kurniawan, R., Habibie, M. N., & Permana, D. S. (2012). KAJIAN DAERAH BARISAN GELOMBANG TINGGI DI PERAIRAN INDONESIA. *Jurnal Meteorologi dan Geofisika*, *13*(3). https://doi.org/10.31172/jmg.v13i3.135

Kusmardiyanti, R., & Yusuf, M. (2022). *Studi Pengaruh SuhuPermukaan Laut Di* Selat Makassar *Terhadap Intensitas Curah Hujan Kota Balikpapan*. *5*, 7.

Labania, H. M., M.S, S., & Khakhim, N. (2019). *Kajian Spasial Variabilitas Karakteristik Gelombang Laut Di Perairan* Selat Makassar. Universitas Gadjah Mada.

Listriani, D., Setyaningrum, A. H., & Eka, F. (2018). Penerapan Metode Asosiasi Menggunakan Algoritma Apriori Pada Aplikasi Analisa Pola Belanja Konsumen (Studi Kasus Toko Buku Gramedia Bintaro). Jurnal Teknik Informatika, *9*(2). https://doi.org/10.15408/jti.v9i2.5602

Mahyudin, M., Suprayogi, I., & Trimaijon, T. (2014). Model Prediksi Liku Kalibrasi Menggunakan Pendekatan Jaringan Saraf Tiruan (ZST) (Studi Kasus: Sub DAS Siak Hulu) (Doctoral dissertation, Riau University).

Malisan, J. (2019). Rendahnya Manajemen Keselamatan Pelayaran Pada Perairan Alur Laut Kepulauan Indonesia (Alki) II Dan III. *Warta Penelitian Perhubungan*, *26*(2), 81. https://doi.org/10.25104/warlit.v26i2.868

Manurung, E. R. S., Adytia, D., & Subasita, N. (2020). Wave Prediction by using *Support* Vector Regression, Study Case in Jakarta Bay. *2020 8th International Conference on Information and Communication Technology (ICoICT)*, 1–5. https://doi.org/10.1109/ICoICT49345.2020.9166297

Nikoo, M. R., & Kerachian, R. (2017). Wave Height Prediction Using Artificial Immune Recognition Systems (AIRS) and Some Other Data Mining Techniques. *Iranian Journal of Science and Technology, Transactions of Civil Engineering*, *41*(3), 329–344. https://doi.org/10.1007/s40996-017-0067-y

Nofriansyah, D. (2014). *Konsep Data Mining Vs Sistem Pendukung Keputusan*. deepublish.

Prasetiyo, B., Rifa'i, A., Diastiara, I. R., Indriyani, L., & Putro, W. P. (2017). *Pembuatan Monitoring Kecepatan Angin Dan Arah Angin Menggunakan Mikrokontroler Arduino*. 9.

Pratama, O. (2020, July 1). Konservasi Perairan Sebagai Upaya menjaga Potensi Kelautan dan Perikanan Indonesia. *Https://Kkp.Go.Id/*. https://kkp.go.id/djprl/artikel/21045-konservasi-perairan-sebagai-upaya-menjaga-potensi-kelautan-dan-perikanan-indonesia

Priyana, F. A., & Kardianawati, A. (2015). Data Mining Asosiasi Untuk Menentukan Cross-Selling Produk Menggunakan Algoritma Frequent Pattern-Growth Pada Koperasi Karyawan Pt. Phapros Semarang.

Putu, I., & Setyawan, A. R. (2020). Pengaruh Angin Monsun Di Laut Selatan Bali Terhadap Keselamatan Pelayaran (Periode Data 1996-2016). Universitas Maritim Amni (Unimar Amni).

Rahma, A., & Purnomo, I. W. A. (2018, June 13). Kapal Tenggelam di Selat Makassar, 13 Penumpang Tewas. *Tempo.Co*. https://nasional.tempo.co/read/1097961/kapal-tenggelam-di-selat-makassar-13-penumpang-tewas

Reonaldho, J. V., Saepudin, D., & Adytia, D. (2020). *Prediksi Gelombang Ekstrim Air Laut di Pelabuhan Tanjung Priok Menggunakan Algoritma ID3*. 14.

Ridwan, & Desha Sanjaya. (2022). Analisis Faktor – Faktor Yang Mempengaruhi Kecelakaan Kapal Pada Alur Pelayaran Di Pelabuhan Kelas Iii Juwana. *Repository Universitas Maritim Amni (Unimar Amni) Semarang*.

Rifqi, M. R., Setiawan, B. D., & Bacthiar, F. A. (2018). *Support Vector Regression Untuk Peramalan Permintaan Darah: Studi Kasus Unit Transfusi Darah Cabang – PMI Kota Malang*. 11.

Rustam, I. (2016). Tantangan ALKI dalam Mewujudkan Cita-cita Indonesia sebagai Poros Maritim Dunia. *Indonesian Perspective*, *1*(1), 1–21. https://doi.org/10.14710/ip.v1i1.10426

Samuel, D. (2008). Penerapan Stuktur *FP-tree* dan Algoritma *FP-Growth* dalam Optimasi Penentuan Frequent Itemset. *Institut Teknologi Bandung*, 6.

Sucipto. (2020, Oktober). Kapal Tenggelam, Enam ABK Terombang-ambing 26 Jam di Selat Makassar. *Kompas*. https://www.kompas.id/baca/nusantara/2020/10/19/kapal-tenggelam-enam-abk-terombang-ambing-26-jam-di-selat-makassar

Suntoro, J. (2018). *Data Mining Algoritme dan Implementasi Menggunakan Bahasa Pemrograman PHP*.

Syaefudin, A. (2017, November 13). Kapal Nelayan Asal Pati Terbalik di Selat Makasar. *DetikNews*. https://news.detik.com/berita-jawa-tengah/d-3725529/kapal-nelayan-asal-pati-terbalik-di-selat-makasar

Tabler, R. D. (1994). *Design guidelines for the control of blowing and drifting snow* (No. SHRP-H-381). Washington, DC: Strategic Highway Research Program, National Research Council.

LAMPIRAN

Lampiran 1 Dataset dan Hasil

https://drive.google.com/drive/folders/1KFx3toX8NeMQ-tChHpA94cGpYSrm8V-T?usp=sharing

Lampiran 2 *Source Code*

a. Data Integration

```
import seaborn as sns
import pandas as pd
from google.colab import data_table
data_table.enable_dataframe_formatter()
df1= pd.read_csv('bmkg_utara_siap_integration.csv')
df2= pd.read_csv('remss_utara_siap_integration.csv')
df3= pd.read_csv('brin_utara_siap_integration.csv')
result = pd.merge(pd.merge(df1,df2,on=['tanggal','latitude','lon
gitude'], how='outer'),df3,on=['tanggal','latitude','longitude']
,how='outer')
result.to_csv('dataset_integration_utara_outer.csv',index=False)
```

b. Data Transformation (Numerik ke Kategori)

```
import pandas as pd
df= pd.read_csv('categorical_utara.csv')
df['kelompok tinggi gelombang'] = pd.cut(df['tinggi gelombang'],
bins = [-1,0.5, 1.25, 2.50, 4, 6, 9, 14],
labels = ['gelombang tenang','gelombang rendah','gelombang sedan
g', 'gelombang tinggi', 'gelombang sangat tinggi', 'gelombang ek
strem', 'gelombang sangat ekstrem'])
df['kelompok curah hujan'] = pd.cut(df['curah hujan'], bins = [-
1,0, 20, 50, 100, 500],labels = ['tidak hujan','hujan ringan', '
hujan sedang', 'hujan lebat', 'hujan sangat lebat'])
df['kelompok arah angin'] = pd.cut(df['arah angin'],bins = [0, 1
2, 34, 57, 79, 102, 124, 147, 169, 192, 214, 237, 259, 282, 304,
 327, 349, 361],labels = ['U', 'UTL', 'TL', 'TTL','T', 'TTG', 'T
G', 'STG', 'S', 'SBD', 'BD', 'BBD', 'B', 'BBL', 'BL', 'UBL', 'U'
],ordered=False)
df['kelompok kecepatan angin'] = pd.cut(df['kecepatan angin'],
bins = [0, 0.3, 1.5, 3.3, 5.5, 8, 10.8, 13.9, 17.2, 20.7, 24.5,
28.4, 32.6, 100],labels = ['angin tenang', 'angin sedikit tenang
', 'sedikit hembusan angin', 'hembusan angin pelan', 'hembusan a
ngin sedang', 'angin sejuk', 'hembusan angin kuat', 'angin mende
kati kencang', 'angin kencang', 'angin kencang sekali', 'angin b
adai', 'angin badai dahsyat','angin badai topan'])
```

```
df['kelompok suhu permukaan laut'] = pd.cut(df['suhu permukaanlaut'],bins = [22.6, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5],
labels = ['hijau tua', 'hijau biasa', 'hijau muda', 'kuning', 'orange', 'merah', 'coklat muda', 'coklat tua'])
select_coloumn = df[['kelompok tinggi gelombang','kelompok curah hujan','kelompok arah angin','kelompok kecepatan angin','kelompok suhu permukaan laut']]
select_coloumn.to_csv('categorical_utara.csv',index=False)
```

c. *Binning*

```
import pandas as pd
df = pd.read_csv('categorical_utara.csv')
df['kelompok angin'] = df['kelompok arah angin'].str.cat(df['kelompok kecepatan angin'], sep=' - ')
dataset = df[['kelompok angin','kelompok curah hujan','kelompok suhu permukaan laut','kelompok tinggi gelombang']]
dataset.to_csv('dataset_siap_association_utara_with_binning.csv', index=False)
```

d. Membuat header itemset

```
import pandas as pd
import collections
#menstel agar semua coloumn terlihat
pd.set_option('display.max_columns', None)
#menstel agar semua baris terlihat
pd.set_option('display.max_rows', None)
dataset = pd.read_csv('dataset_siap_association_utara_after_handling_missing_value_with_binning.csv')
data_array = dataset.to_numpy()
#menggunakan set karena set tidak beraturan dan tidak dapat diubah isinya. set juga digunakan untuk mencari nilai unique
item_unik = set()
for unik in data_array:
    for item in unik:
        item_unik.add(item)
kolom_ = sorted(item_unik)
kolom_item_unik = {}
for col_idx, item in enumerate(kolom_):
    kolom_item_unik[item] = col_idx
print(kolom_item_unik)
print(item_unik)
```

```
df_kolom_item_unik = pd.DataFrame(kolom_item_unik, index = [0])
print(df_kolom_item_unik)
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
#mengubah data array ke dataframe true or false
te = TransactionEncoder()
te_ary = te.fit(data_array).transform(data_array)
data_encoding = pd.DataFrame(te_ary, columns=te.columns_)
data_encoding
use_colnames = True
colname_map = None
if use_colnames:
    colname_map = {idx: item for idx, item in enumerate(data_encod
    ing.columns)}
print(colname_map)
num_itemsets = len(data_encoding.index)
is_sparse = False
if hasattr(data_encoding, 'sparse'):
    if data_encoding.size == 0:
        itemsets = data_encoding.values
    else:
        itemsets = data_encoding.sparse.to_coo().tocsr()
        is_sparse = True
else:
    itemsets = data_encoding.values
print(itemsets)
jumlah_baris_dataset = len(data_encoding.index)
count_support = np.array(np.sum(itemsets, axis = 0) / float(jumlah
_baris_dataset))
print(count_support)
item = []
for col in data_encoding:
  item.append(col)
support_count = pd.DataFrame(count_support, columns=['support_coun
t'])
item = pd.DataFrame(item, columns=['item'])
df_all_rows = pd.concat([item, support_count], axis =1 )
df_support_count = df_all_rows.sort_values(['support_count'],ascen
ding=False)
df_support_count
df_support_count.to_excel('support_count_utara_fix.xlsx')
```

```
min_support = 0.003
select_frequent_item_set = df_support_count[
               (df_support_count['support_count'] >= min_support )]
select_frequent_item_set.rename(columns={"support_count": "frequent
itemset"}, inplace=True)
# select_frequent_item_set.to_excel('frequent_item_set_utara.xlsx')
# len(select_frequent_item_set)
select_frequent_item_set= pd.read_excel('frequent_item_set_utara.xls
x')
frequent_item_set = select_frequent_item_set['item'].to_numpy()
frequent_item_set
conditional_pattern = []
for ab in data_array:
  conditional_pattern_itemset = []
  for cd in ab:
    if cd in frequent_item_set:
      conditional_pattern_itemset.append(cd)
  conditional_pattern.append(conditional_pattern_itemset)
conditional_pattern
conditional_pattern_fix = pd.DataFrame(conditional_pattern)
conditional_pattern_fix
from collections import Counter
def solve(arr1, arr2):
    # Our output array
    res = []
    # Counting Frequency of each
    # number in arr1
    f = Counter(arr1)
    # Iterate over arr2 and append all
    # occurrences of element of
    # arr2 from arr1
    for e in arr2:
        # Appending element 'e',
        # f[e] number of times
        res.extend([e]*f[e])
        # Count of 'e' after appending is zero
        f[e] = 0
    # Remaining numbers in arr1 in sorted
    # order (Numbers with non-zero frequency)
    rem = list(sorted(filter(
      lambda x: f[x] != 0, f.keys())))
    # Append them also
    for e in rem:
        res.extend([e]*f[e])
    return res
```

```
if __name__=='__main__':
    a = []
    arr1 = conditional_pattern
    arr2 = frequent_item_set
    for i in range(len(arr1)):
        b = solve(arr1[i], arr2)
        a.append(b)
    print(a)
filter(lambda v: v is not None, a)
df_conditional_pattern = pd.DataFrame(a)
df_conditional_pattern
df_conditional_pattern.to_csv('sorted_items_utara_fix_sekali2.csv',i
ndex=False)
df_conditional_pattern.to_excel('sorted_items_selatan.xlsx',index=Fa
lse)
df_conditional_pattern['frequent items (ordered)'] = df_conditional_
pattern.apply(
    lambda x: ','.join(x.dropna().astype(str)),
    axis=1
)
df_conditional_pattern
df_conditional_pattern = df_conditional_pattern[["frequent items (or
dered)"]]
df_conditional_pattern
df_items = df_conditional_pattern["frequent items (ordered)"]
itemset = df_items.to_numpy()
itemset
itemset_fix = pd.DataFrame(itemset)
itemset_fix
itemset_fix.to_csv('sorted_items_tengah.csv')
itemset_fix.to_excel('sorted_items_tengah.xlsx')
```

e. Algoritma FP-Growth

```
import pandas as pd
import collections
from mlxtend.preprocessing import TransactionEncoder
#menstel agar semua coloumn terlihat
pd.set_option('display.max_columns', None)
#menstel agar semua baris terlihat
pd.set_option('display.max_rows', None)
dataset = pd.read_csv('sorted_items_selatan_fix_sekali2.csv')
data_array = [[j for j in i if not pd.isnull(j)] for i in dataset.va
lues]
#menggunakan set karena set tidak beraturan dan tidak dapat diubah i
sinya. set juga digunakan untuk mencari nilai unique
item_unik = set()
for unik in data_array:
    for item in unik:
        item_unik.add(item)
kolom_ = sorted(item_unik)
kolom_item_unik = {}
for col_idx, item in enumerate(kolom_):
    kolom_item_unik[item] = col_idx
print(kolom_item_unik)
print(item_unik)
df_kolom_item_unik = pd.DataFrame(kolom_item_unik, index = [0])
print(df_kolom_item_unik)
#mengubah data array ke dataframe true or false
te = TransactionEncoder()
te_ary = te.fit_transform(data_array)
data_encoding = pd.DataFrame(te_ary, columns=te.columns_)
use_colnames = True
colname_map = None
if use_colnames:
    colname_map = {idx: item for idx, item in enumerate(data_encodin
g.columns)}
print(colname_map)
num_itemsets = len(data_encoding.index)
is_sparse = False
if hasattr(data_encoding, 'sparse'):
    if data_encoding.size == 0:
        itemsets = data_encoding.values
    else:
        itemsets = data_encoding.sparse.to_coo().tocsr()
        is_sparse = True
else:
    itemsets = data_encoding.values
print(itemsets)
```

```
jumlah_baris_dataset = len(data_encoding.index)
count_support = np.array(np.sum(itemsets, axis = 0) / float(jumlah_b
aris_dataset))
print(count_support)
import pandas as pd
item = []
for col in data_encoding:
  item.append(col)
support_count = pd.DataFrame(count_support, columns=['support_count'
])
item = pd.DataFrame(item, columns=['item'])
df_all_rows = pd.concat([item, support_count], axis =1 )
min_support = 0.003
max_len = None
if min_support <= 0.:
    raise ValueError('min support must be positif and the number bet
ween the interval 0-1. Got %s.' %min_support)
items_ = np.nonzero(count_support >= min_support)[0] #untuk mencari
support yang lebih dari minimal support
print(items_) #print itemnya
indices = count_support[items_].argsort()[::-1] #
rank = {item: i for i, item in enumerate(items_[indices])}  #di rank
 dari terbawah ke tertinggi
print(indices) #urutan index dari item
print(rank) #di rank dari terbawah ke tertinggi
def create_FPNode(item, count = 0, parent = None):
  node = {
  'item': item,
  'count': count,
  'parent': parent,
  'children': {}}
  if parent is not None:
      parent['children']['item'] = node
  return  node
def create_FPtree(rank = None):
  root = create_FPNode(None)
  nodes = collections.defaultdict(list)
  cond_items = []
  tree = {
      'root': root,
      'nodes': nodes,
      'cond_items': cond_items,
      'rank': rank
  }
  return tree
```

```
import pprint
def insert_itemset(tree, itemset, count = 1):
  tree['root']['count'] += count
  index = 0
  node = tree['root']
  for item in itemset:
      if item in node['children']:
          child = node['children'][item]
          child['count'] += count
          node = child
          index += 1
      else:
          break
  for item in itemset[index:]:
      child_node = create_FPNode(item, count, node)
      tree['nodes'][item].append(child_node)
      node = child_node
  pprint.pprint(node)
tree = create_FPtree(rank)
for i in range(num_itemsets):
    if is_sparse:
        nonnull = itemsets.indices[itemsets.indptr[i]:itemsets.indpt
r[i+1]]
    else:
        nonnull = np.where(itemsets[i, :])[0]
    itemset = [item for item in nonnull if item in rank]
    itemset.sort(key=rank.get, reverse=True)
    insert_itemset(tree, itemset)
import math
import itertools
def fpg_step(tree, minsup, colnames, max_len):
  count = 0
  items = tree['nodes'].keys()

  is_path = True
  if len(tree['root']['children']) >1:
      is_path = False

  for i in tree['nodes']:
      if len(tree['nodes'][i]) >1 or len(tree['nodes'][i][0]['childr
en']) >1:
          is_path = False
```

```
   if is_path:
       size_remain = len(items) + 1
       if max_len:
           size_remain = max_len - len(tree['cond_items']) + 1
       for i in range(1, size_remain):
           for itemset in itertools.combinations(items, i):
               count += 1
               support = min([tree['nodes'][i][0]['count'] for i in ite
mset])
               yield support, tree['cond_items'] + list(itemset)
   elif not max_len or max_len > len(['cond_items']):
       for item in items:
           count += 1
           support = sum([node['count'] for node in tree['nodes'][item]
])
           yield support, tree['cond_items'] + [item]
   if not is_path and (not max_len or max_len > len(tree['cond_items'])
):
       for cond_item in items:
           branches = []
           count = collections.defaultdict(int)
           for node in tree['nodes'][cond_item]:
               path = []
               if item is None:
                   return path
               node_ = node['parent']
               while node_['item'] is not None:
                   path.append(node_['item'])
                   node_ = node_['parent']
               path.reverse()
               branch = path
               branches.append(branch)
               for item in branch:
                   count[item] += node['count']
           print(path)
           items_cond = [item for item in count if count[item] >= minsup]
           items_cond.sort(key=count.get)
           rank = {item: i for i, item in enumerate(items)}
           cond_tree = create_FPtree(rank)
           for idx, branch in enumerate(branches):
               branch = sorted([i for i in branch if i in rank],
                               key=rank.get, reverse=True)
               insert_itemset(cond_tree, branch, tree['nodes'][cond_item]
[idx]['count'])
           cond_tree['cond_items'] = tree['cond_items'] + [cond_item]
           for sup, iset in fpg_step(cond_tree, minsup, colnames, max_len
):
               yield sup, iset
```

```
minsup = math.ceil(min_support * len(data_encoding.index))
generator = fpg_step(tree, minsup, colname_map, max_len)
generator
itemsets = []
supports = []
for sup, iset in generator:
    itemsets.append(frozenset(iset))
    supports.append(sup / num_itemsets)

res_df = pd.DataFrame({'support': supports, 'itemsets': itemsets})
if colname_map is not None:
    res_df['itemsets'] = res_df['itemsets'].apply(lambda x: frozense
t([colname_map[i] for i in x]))
res_df = res_df[res_df.support >= min_support]
res_df_sort = res_df.sort_values(['support'],ascending=False)
def conviction(sAC, sA, sC):
  confidence = sAC/sA
  conviction = np.empty(confidence.shape, dtype=float)
  if not len(conviction.shape):
      conviction = conviction[np.newaxis]
      confidence = confidence[np.newaxis]
      sAC = sAC[np.newaxis]
      sA = sA[np.newaxis]
      sC = sC[np.newaxis]
  conviction[:] = np.inf
  conviction[confidence < 1.] = ((1. - sC[confidence < 1.]) /(1. - c
 onfidence[confidence < 1.]))
  return conviction
metric_dict = {
        "antecedent support": lambda _, sA, __: sA,
        "consequent support": lambda _, __, sC: sC,
        "support": lambda sAC, _, __: sAC,
        "confidence": lambda sAC, sA, _: sAC/sA,
        "lift": lambda sAC, sA, sC: metric_dict["confidence"](sAC, s
                A, sC)/sC,
        "leverage": lambda sAC, sA, sC: metric_dict["support"](
             sAC, sA, sC) - sA*sC,
        "conviction": lambda sAC, sA, sC: conviction(sAC, sA, sC)}
columns_ordered = ["antecedent support",
                   "consequent support",
                   "support",
                   "confidence",
                   "lift",
                   "leverage",
                   "conviction"]
```

```
if not res_df.shape[0]:
      raise ValueError('The input DataFrame `df` containing '
                       'the frequent itemsets is empty.')
if not all(col in res_df.columns for col in ["support", "itemsets"]):
      raise ValueError("Dataframe needs to contain the\
                       columns 'support' and 'itemsets'")
metric = 'confidence'
support_only = False
if support_only:
    metric = 'support'
else:
  if metric not in metric_dict.keys():
      raise ValueError("Metric must be 'confidence' or 'lift', got '{}'"
                       .format(metric))
keys = res_df['itemsets'].values
values = res_df['support'].values
frozenset_vect = np.vectorize(lambda x: frozenset(x))
frequent_items_dict = dict(zip(frozenset_vect(keys), values))
print(keys)
print(values)
print(frequent_items_dict)
from itertools import combinations
min_threshold = 0.98
rule_antecedents = []
rule_consequents = []
rule_supports = []
for k in frequent_items_dict.keys():
  sAC = frequent_items_dict[k]
  # to find all possible combinations
  for idx in range(len(k)-1, 0, -1):
    # of antecedent and consequent
    for c in combinations(k, r=idx):
      antecedent = frozenset(c)
      consequent = k.difference(antecedent)
      if support_only:
        # support doesn't need these,
        # hence, placeholders should suffice
        sA = None
        sC = None
      else:
        try:
          sA = frequent_items_dict[antecedent]
          sC = frequent_items_dict[consequent]
        except KeyError as e:
```

```
            s = (str(e) + 'You are likely getting this error'
                          ' because the DataFrame is missing '
                          ' antecedent and/or consequent '
                          ' information.'
                          ' You can try using the '
                          ' `support_only=True` option')
            raise KeyError(s)
          # check for the threshold
        score = metric_dict[metric](sAC, sA, sC)
        if score >= min_threshold:
            rule_antecedents.append(antecedent)
            rule_consequents.append(consequent)
            rule_supports.append([sAC, sA, sC])
if not rule_supports:
  pd.DataFrame(columns=["antecedents", "consequents"] + columns_ordered)
else:
  # generate metrics
  rule_supports = np.array(rule_supports).T.astype(float)
  df_res = pd.DataFrame(
    data=list(zip(rule_antecedents, rule_consequents)),
    columns=["antecedents", "consequents"])
  if support_only:
    sAC = rule_supports[0]
    for m in columns_ordered:
        df_res[m] = np.nan
    df_res['support'] = sAC
  else:
    sAC = rule_supports[0]
    sA = rule_supports[1]
    sC = rule_supports[2]
    for m in columns_ordered:
      df_res[m] = metric_dict[m](sAC, sA, sC)
df_res= df_res.sort_values(['lift','confidence','support'],ascending=Fals
e)
df_res
# filtering the rows where consequent is Gelombang
contain_values = df_res[df_res['consequents'].astype(str).str.contains('g
elombang tinggi)]
contain_values
contain_values.to_excel('hasil_asosiasi_utara_gelombang_tinggi.xlsx', ind
ex=False)
```

f. Prediksi Menggunakan SVR

```
import pandas as pd
import numpy as np
import collections
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
#menstel agar semua coloumn terlihat
pd.set_option('display.max_columns', None)
#menstel agar semua baris terlihat
pd.set_option('display.max_rows', None)
dataset = pd.read_excel('data gabung.xlsx')
data_tinggi_gelombang = dataset[['tinggi gelombang']]
#menentukan paramateter prediktor
data_normalisasi = dataset[['kecepatan angin','arah angin','tinggi
gelombang']].to_numpy()
#normalisasi variabel
min_max_scaler = preprocessing.MinMaxScaler(feature_range=(0,1))
#inisialisasi normalisasi MinMax
scaler_dataset = min_max_scaler.fit_transform(data_normalisasi)
#transformasi MinMax untuk fitur
print("Normalisasi variabel:")
print(scaler_dataset)
#mengambil data prediktor
X = scaler_dataset[:, :-1]
#mengambil data target
Y = scaler_dataset[:, -1:]
np.reshape(Y, (-1,1))
#Split Dataset into Train and Test
X_train, X_test, y_train, y_test = train_test_split(X,Y,
train_size=0.8,shuffle=False)
print(X_train.shape), print(y_train.shape)
print(X_test.shape), print(y_test.shape)
aktual_gelombang_train = pd.DataFrame(y_train)
aktual_gelombang_train.rename(columns={0: "aktual gelombang laut train"},
inplace=True)
aktual_gelombang_train
aktual_gelombang_test = pd.DataFrame(y_test)
aktual_gelombang_test.rename(columns={0: "aktual gelombang laut test"},
inplace=True)
```

```
#Menentukan Kernel

model = SVR(kernel = 'rbf')
model = model.fit(X_train, y_train)
#Mencari Parameter Kernel Terbaik
params = {
    'C':[0.0001,0.001, 0.1, 1,5,5, 10,20,50,100,1000],
    'epsilon': [0.0001,0.001,0.01,0.05,0.1,10,20,50,100,1000],
    'gamma': [0.01,0.1,1,0.5,10,'scale','auto']}

model_grid_search  =  GridSearchCV(estimator=model,  param_grid=params,
cv=5,
n_jobs=-1, verbose=1,refit=True)
model_grid_search.fit(X_train, y_train)

#Memanggil parameter terbaik
print('param_terbaik adalah = ',model_grid_search.best_params_)

#Print Hasil Rank Test Score
result = pd.DataFrame(model_grid_search.cv_results_)
result = result.sort_values('rank_test_score')
# #Membuat model SVR
# best_param2 = model_grid_search.best_params_



model_svr = SVR(kernel='rbf', C=0.1, epsilon=0.05, gamma='scale')
#      model_svr      =      SVR(kernel='rbf',      C=best_param2['C'],
epsilon=best_param2['epsilon'], gamma=best_param2['gamma'])
model_svr.fit(X_train, y_train)

result_df = pd.DataFrame([],columns=[])
#Memprediksi Data Train
y_svr_train= model_svr.predict(X_train)
result_df['svr_predicted_train'] = y_svr_train # SVR
prediksi_train = result_df['svr_predicted_train']
hasil_prediksi_train = result_df.to_numpy()
hasil_prediksi_train = pd.DataFrame(hasil_prediksi_train)
hasil_prediksi_train.rename(columns={0:"prediksi gelombang"},
inplace=True)
hasil_prediksi_train
```

```
angin = pd.DataFrame(X_train, columns = ['kecepatan angin','arah angin'])
prediksi_gelombang_train = pd.DataFrame(hasil_prediksi_train, columns =
['prediksi gelombang'])
aktual_gelombang_train = pd.DataFrame(aktual_gelombang_train, columns =
['aktual gelombang laut train'])
join_dataframe_prediksi_train = pd.concat([angin,
prediksi_gelombang_train], axis=1)
join_dataframe_aktual_train = pd.concat([angin, aktual_gelombang_train],
axis=1)
join_prediksi_train = join_dataframe_prediksi_train.to_numpy()

join_aktual_train = join_dataframe_aktual_train .to_numpy()

unscaled_prediksi_train = min_max_scaler.inverse_transform
(join_prediksi_train)
unscaled_aktual_train = min_max_scaler.inverse_transform
(join_aktual_train)

hasil_denormalisasi_prediksi_train = pd.DataFrame
(unscaled_prediksi_train)
hasil_denormalisasi_prediksi_train.rename(columns={0: "kecepatan angin"},
inplace=True)
hasil_denormalisasi_prediksi_train.rename(columns={1: "arah angin"},
inplace=True)
hasil_denormalisasi_prediksi_train.rename(columns={2: "prediksi gelombang
laut"}, inplace=True)
select_prediksi_gelombang_train =
hasil_denormalisasi_prediksi_train[['prediksi gelombang laut']]
hasil_denormalisasi_aktual_train = pd.DataFrame(unscaled_aktual_train)
hasil_denormalisasi_aktual_train.rename(columns={0: "kecepatan angin"},
inplace=True)
hasil_denormalisasi_aktual_train.rename(columns={1: "arah angin"},
inplace=True)
hasil_denormalisasi_aktual_train.rename(columns={2: "aktual gelombang
laut"}, inplace=True)
select_aktual_gelombang_train = hasil_denormalisasi_aktual_train[['aktual
gelombang laut']]
```

```
hasil_denormalisasi_aktual_train = pd.DataFrame(unscaled_aktual_train)
hasil_denormalisasi_aktual_train.rename(columns={0: "kecepatan angin"},
inplace=True)
hasil_denormalisasi_aktual_train.rename(columns={1: "arah angin"},
inplace=True)
# hasil_denormalisasi_aktual_train.rename(columns={2: "suhu permukaan
laut"}, inplace=True)
# hasil_denormalisasi_aktual_train.rename(columns={3: "curah hujan"},
inplace=True)
hasil_denormalisasi_aktual_train.rename(columns={2: "aktual gelombang
laut"}, inplace=True)
select_aktual_gelombang_train = hasil_denormalisasi_aktual_train
[['aktual gelombang laut']]

result =  pd.concat([select_prediksi_gelombang_train,
select_aktual_gelombang_train],axis=1)
result['svr_error_test'] = abs(result['aktual gelombang laut'] -
result['prediksi gelombang laut'])
svr_error_test = result['svr_error_test']

aktual = result['aktual gelombang laut']
prediksi = result['prediksi gelombang laut']

#Visualisasi Hasil Prediksi Data Train
from matplotlib import pyplot as plt
plt.plot(aktual, 'r--', label='Aktual_Train')
plt.plot(prediksi, 'b', label='Prediksi_Train')
plt.title('Grafik Data Aktual dan Prediksi Train')
plt.legend()
plt.rcParams["figure.figsize"] = (50,3)
plt.show()

#Save Model
import pickle
pickle.dump(model_svr, open('model_svr.sav', 'wb'))
loaded_model = pickle.load(open('model_svr.sav', 'rb'))
result_df2 = pd.DataFrame([],columns=[])

y_svr_test = loaded_model.predict(X_test) # SVR
result_df2['svr_predicted_test'] = y_svr_test
prediksi_test = result_df2['svr_predicted_test']
hasil_prediksi_test = result_df2.to_numpy()
hasil_prediksi_test
```

```
angin_test = pd.DataFrame(X_test, columns = ['kecepatan angin','arah
angin'])
prediksi_gelombang_test = pd.DataFrame(hasil_prediksi_test, columns =
['prediksi gelombang'])
aktual_gelombang_test = pd.DataFrame(aktual_gelombang_test, columns =
['aktual gelombang laut test'])
join_dataframe_prediksi_test = pd.concat([angin_test,
prediksi_gelombang_test], axis=1)
join_dataframe_aktual_test = pd.concat([angin_test,
aktual_gelombang_test], axis=1)
join_prediksi_test = join_dataframe_prediksi_test.to_numpy()
join_prediksi_test

join_aktual_test = join_dataframe_aktual_test .to_numpy()

unscaled_prediksi_test =
min_max_scaler.inverse_transform(join_prediksi_test)

unscaled_aktual_test = min_max_scaler.inverse_transform(join_aktual_test)

hasil_denormalisasi_prediksi_test = pd.DataFrame(unscaled_prediksi_test)
hasil_denormalisasi_prediksi_test.rename(columns={0: "kecepatan angin"},
inplace=True)
hasil_denormalisasi_prediksi_test.rename(columns={1: "arah angin"},
inplace=True)
hasil_denormalisasi_prediksi_test.rename(columns={2: "prediksi gelombang
laut"}, inplace=True)
select_prediksi_gelombang_test =
hasil_denormalisasi_prediksi_test[['prediksi gelombang laut']]
select_prediksi_gelombang_test
hasil_denormalisasi_aktual_test = pd.DataFrame(unscaled_aktual_test)
hasil_denormalisasi_aktual_test.rename(columns={0: "kecepatan angin"},
inplace=True)
hasil_denormalisasi_aktual_test.rename(columns={1: "arah angin"},
inplace=True)
hasil_denormalisasi_aktual_test.rename(columns={2: "aktual gelombang
laut"}, inplace=True)
select_aktual_gelombang_test = hasil_denormalisasi_aktual_test[['aktual
gelombang laut']]
select_aktual_gelombang_test
```

```
result_test =  pd.concat([select_prediksi_gelombang_test,
select_aktual_gelombang_test],axis=1)
result_test['svr_error_test'] = abs(result_test['aktual gelombang laut']
- result_test['prediksi gelombang laut'])
svr_error_test = result_test['svr_error_test']
# result_test.to_excel('hasil prediksi denormalisasi 4 param
tengah.xlsx',index=False)

aktual_test = result_test['aktual gelombang laut']
prediksi_test = result_test['prediksi gelombang laut']
# result_test.to_excel('testing_tengah_2 param.xlsx',index=False)

#Visualisasi Data Testing
from matplotlib import pyplot as plt
plt.plot(aktual_test, 'r--', label='Aktual_Testing')
plt.plot(prediksi_test, 'b', label='Prediksi_Testing')
plt.title('Grafik Data Aktual dan Prediksi')
plt.legend()
plt.rcParams["figure.figsize"] = (50,3)
plt.show()

#Menghitung REMSE Data Training dan Data Testing
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_percentage_error
import math
from math import *
rmse_train = mean_squared_error(y_train, y_svr_train, squared=False)
print("Training Set Root Mean Absolute Error (RMSE): %.4f" % rmse_train)
rmse_test = mean_squared_error(y_test, y_svr_test, squared=False)
print("Testing Set Root Mean Absolute Error (RMSE): %.4f" % rmse_test)
import pickle
pickle.dump(loaded_model, open('model_svr.sav', 'wb'))

data_train = pd.read_excel('data gabung.xlsx')

data_actual_gelombang = data_train[['tinggi gelombang']]
data_actual_gelombang.rename(columns={"tinggi gelombang": "aktual
gelombang laut"}, inplace=True)

data_train_numpy = data_train[['kecepatan angin','arah angin', 'tinggi
gelombang']].to_numpy()
```

```
min_max_scaler_train = preprocessing.MinMaxScaler(feature_range=(0,1))
#inisialisasi normalisasi MinMax
scaler_data_train = min_max_scaler_train.fit_transform(data_train_numpy)
#transformasi MinMax untuk fitur
print("Normalisasi Data Train :")
print(scaler_data_train)

prediktor_train = scaler_data_train[:, :-1]
target_train =  scaler_data_train[:, -1:]

#Menentukan Kernel

model_new = SVR(kernel = 'rbf')
model_new = model_new.fit(prediktor_train, target_train)

#Mencari Parameter Kernel Terbaik
params = {
    'C':[0.001,0.01,0.1,0,1,10,100,1000],
    'epsilon': [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000],
    'gamma': [0.1, 1,10,'scale','auto',100]}

model_grid_search = GridSearchCV(estimator=model_new, param_grid=params,
cv=5,
n_jobs=-1, verbose=1,refit=True)
model_grid_search.fit(prediktor_train, target_train)

#Memanggil parameter terbaik
print('param_terbaik adalah = ',model_grid_search.best_params_)
best_param_new = model_grid_search.best_params_
model_svr_new = SVR(kernel='rbf', C=best_param_new['C'],
epsilon=best_param_new['epsilon'], gamma=best_param_new['gamma'])
model_svr_new.fit(prediktor_train, target_train)

#Save Model
import pickle
pickle.dump(model_svr_new, open('model_svr.sav', 'wb'))

data_testing_new = pd.read_excel('januari.xlsx')
data_testing_new
```

```
data_aktual_gelombang_test_new = data_testing_new[['tinggi gelombang']]
data_aktual_gelombang_test_new

#Prediksi Data Testing
data_test_numpy = data_testing_new[['kecepatan angin','arah angin',
'tinggi gelombang']].to_numpy()
data_test_numpy

from sklearn.preprocessing import MinMaxScaler
min_max_scaler_test = preprocessing.MinMaxScaler(feature_range=(0,1))
#inisialisasi normalisasi MinMax
scaler_data_test = min_max_scaler_test.fit_transform(data_test_numpy)
#transformasi MinMax untuk fitur
print("Normalisasi Data Test :")
print(scaler_data_test)

prediktor_test = scaler_data_test[:, :-1]
prediktor_test

target_test =  scaler_data_test[:, -1:]
target_test

loaded_model = pickle.load(open('model_svr.sav', 'rb'))
result_prediction_new = pd.DataFrame([],columns=[])

y_svr_test_new = loaded_model.predict(prediktor_test) # SVR
result_prediction_new['svr_predicted_test'] = y_svr_test_new
hasil_prediksi_test_new = result_prediction_new['svr_predicted_test']
hasil_prediksi_test_new  = result_prediction_new.to_numpy()
hasil_prediksi_test_new

angin_test_new = pd.DataFrame(prediktor_test, columns = ['kecepatan
angin','arah angin'])
prediksi_gelombang_test = pd.DataFrame(hasil_prediksi_test_new , columns =
['prediksi gelombang'])
join_dataframe_prediksi_test_new = pd.concat([angin_test_new ,
prediksi_gelombang_test], axis=1)
join_prediksi_test_new = join_dataframe_prediksi_test_new.to_numpy()
join_prediksi_test_new

unscaled_prediksi_test =
min_max_scaler_test.inverse_transform(join_prediksi_test_new)
unscaled_prediksi_test
```

```
hasil_denormalisasi_prediksi_test_new =
pd.DataFrame(unscaled_prediksi_test)
hasil_denormalisasi_prediksi_test_new.rename(columns={0: "kecepatan
angin"}, inplace=True)
hasil_denormalisasi_prediksi_test_new.rename(columns={1: "arah angin"},
inplace=True)
hasil_denormalisasi_prediksi_test_new.rename(columns={2: "prediksi
gelombang laut"}, inplace=True)
select_prediksi_gelombang_test =
hasil_denormalisasi_prediksi_test_new[['prediksi gelombang laut']]
select_prediksi_gelombang_test

prediksi_gelombang_test_new =
pd.DataFrame(select_prediksi_gelombang_test, columns = ['prediksi
gelombang laut'])
aktual_gelombang_test_new = pd.DataFrame(data_aktual_gelombang_test_new,
columns = ['tinggi gelombang'])
result_test_new = pd.concat([prediksi_gelombang_test_new ,
aktual_gelombang_test_new], axis=1)
result_test_new

new = pd.DataFrame()
new = data_train.append(data_testing_new)
new.to_excel('data gabung.xlsx',index=False)
```

LEMBAR PERBAIKAN SKRIPSI

"SISTEM PREDIKSI TINGGI GELOMBANG LAUT DI SELAT MAKASSAR MENGGUNAKAN DATA METEOROLOGI DAN OSEANOGRAFI"

OLEH:

MAGHFIRAH TENRI SUMPALA ZANI
D121181520

Skripsi ini telah dipertahankan pada Ujian Akhir Sarjana tanggal 06 Februari 2023.

Telah dilakukan perbaikan penulisan dan isi skripsi berdasarkan usulan dari penguji dan pembimbing skripsi.

Persetujuan perbaikan oleh tim penguji:

	Nama	Tanda Tangan
Ketua	Prof. Dr. Ir. Indrabayu ST, MT, M.Bus.Sys., IPM, ASEAN. Eng	
Sekretaris	Elly Warni ST.,MT	
Anggota	A. Ais Prayogi Alimuddin ST.,M.Eng	
	Anugrayani Bustamin ST.,MT	

Persetujuan Perbaikan oleh pembimbing:

Pembimbing	Nama	Tanda Tangan
I	Prof. Dr. Ir. Indrabayu ST, MT, M.Bus.Sys., IPM, ASEAN. Eng	
II	Elly Warni ST.,MT	